Universidade de Aveiro
2017

Departamento de Eletrónica, Telecomunicações e Informática

Fernando José Fradique Duarte

Plataforma para Descoberta de Eventos de Interesse Noticioso no Twitter
Platform for the Discovery of Newsworthy Events in Twitter

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Óscar Narciso Mortágua Pereira, Professor auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	Prof. Doutor José Manuel Matos Moreira professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
vogais / examiner committee	Prof. Doutor João Pedro Carvalho Leal Mendes Moreira professor auxiliar do Departamento de Engenharia Informática da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor Óscar Narciso Mortágua Pereira professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos / acknowledgments

Gostaria de agradecer ao professor Óscar Pereira, pelos seus conselhos, apoio e pela disponibilidade que demonstrou durante o decorrer desta dissertação.

palavras-chave Redes Sociais, Twitter, Deteção de eventos, Aprendizagem de Máquina, Programação Dinâmica.

resumo O novo paradigma de comunicação estabelecido pelas Redes Sociais, aliado à sua crescente popularidade no passado recente, contribuíram para suscitar o interesse de diversas áreas de investigação. Uma dessas áreas é a detecção de eventos em Redes Sociais, cuja relevância deriva do seu elevado potencial de aplicabilidade num conjunto diverso de aplicações. Uma dessas aplicações é a deteção de eventos de interesse noticioso em redes Sociais. O objectivo deste trabalho é por isso o de implementar um sistema para deteção de eventos de interesse noticioso no Twitter. Um sistema semelhante proposto na literatura é usado como base desta implementação. Para atingir este propósito foi implementado um algoritmo de segmentação utilizando uma abordagem baseada em programação dinâmica por forma a separar os tweets em segmentos. Um esquema de ponderação tendo em conta o aumento intermitente da frequência dos segmentos, a sua base de suporte em termos de utilizadores e o seu potencial noticioso foi então utilizado para gerar um ranking destes segmentos. A Wikipédia foi utilizada como meio para calcular este potencial noticioso. Os top *K* segmentos neste ranking foram sujeitos a processamento posterior e agrupados em eventos candidatos de acordo com a sua similaridade. Por sua vez estes eventos candidatos foram filtrados por um modelo SVM, treinado em dados anotados manualmente, por forma a reter apenas aqueles relacionados com eventos do mundo real com interesse noticioso. Foi também implementada toda a infra-estrutura de suporte necessária ao sistema, nomeadamente no que diz respeito aos valores pré-calculados considerados necessários ao seu funcionamento. O sistema implementado foi testado com três meses de dados representando um total de 4,770,636 de tweets criados em Portugal e maioritariamente escritos em português. A precisão obtida pelo sistema foi de 76.9 % e a sua sensibilidade de 41.6%.

Keywords Social Media, Twitter, Event Detection, Machine Learning, Dynamic Programming.

Abstract The new communication paradigm established by Social Media, along with its growing popularity in recent years, have contributed to attract an increasing interest by several research fields. One such research field is the field of event detection in Social Media, whose relevance stems from its potential applicability in many diverse applications. One such application is the detection of newsworthy events in Social Media. The purpose of this work is therefore to implement a system to detect newsworthy events in Twitter. A similar system proposed in the literature is used as the base of this implementation. For this purpose a segmentation algorithm was implemented using a dynamic programming approach in order to split the tweets into segments. A weighting scheme that takes into account the burstiness, user support and newsworthiness of the segments was then used to rank these segments. Wikipedia was leveraged in order to derive this newsworthiness. The top *K* segments in this ranking were further processed and clustered into candidate events according to their similarity. These candidate events were then filtered by an SVM model trained on manually annotated data in order to retain only those related to real-world newsworthy events. The support infrastructure required by the system, namely regarding the precomputed values considered necessary to its operation was also implemented. The implemented system was tested with three months of data, representing a total of 4,770,636 tweets created in Portugal and mostly written in the Portuguese language. The precision obtained by the system was 76.9 % with a recall of 41.6%.

Contents

List of Figures

List of Tables

Listings

Acronyms

AOF	Append-Only File
API	Application Programming Interface
BSD	Berkeley Software Distribution
BSON	Binary JSON
CDE	Crime and Disaster related Event
CSV	Comma Separated Values
DAG	Directed Acyclic Graph
EDA	Exploratory Data Analysis
EPG	Electronic Programming Guide
FSD	First Story Detection
GB	Gigabyte
IoT	Internet of Things
JSON	JavaScript Object Notation
LSH	Locality-Sensitive Hashing
M2M	Machine-to-Machine
MB	Megabyte
ML	Machine Learning
MLE	Maximum Likelihood Estimate
MWU	Multi-Word Unit
NER	Named Entity Recognizer
NED	New Event Detection
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NoSQL	Not Only SQL
OSN	Online Social Network
PCA	Principal Component Analysis
POSIX	Portable Operating System Interface
RED	Retrospective Event Detection
SCP	Symmetric Conditional Probability
SSD	Solid State Disk
SVD	Singular Value Decomposition
SVM	Support Vector Machine
UCG	User Generated Content
XML	Extensible Markup Language

1 Introduction

This chapter presents an overall overview of this dissertation. The chapter starts by providing some background context regarding the subject to be addressed by this work. Then the goals as well as the motivation and main contributions are discussed. The outline of the remainder of this document is then presented at the end of the chapter.

1.1 Contextualization

Social Media services have become a very popular medium of communication [1] and users use these services for various different reasons. In the case of Twitter [2,3], a microblogging service, the main reasons found are [4]: **daily chatter**, **conversations**, **sharing information** and **reporting news**. Microblogging services in particular, have become very popular due to their portability, immediacy and ease of use, allowing users to respond and spread information more rapidly [5]. The popularity of these services, their real time nature, the fact that these data reflect aspects of real-world societies and that these data are publicly available, has therefore attracted the attention of researchers in several fields [1,6].

One such field is event detection in Social Media. Event detection has many potential applications, some of which with significant social impact, such as in the detection of natural disasters and to identify and track diseases and epidemics [6]. Another relevant application is the detection of newsworthy events, as events can often be discussed by users in these services before they are even reported in traditional Media [7,8]. In fact this was the topic of the SNOW 2014 Data Challenge [9].

These services however present some challenges, some of which are inherit to their design and usage. Examples of this are the use of informal and abbreviated words, the frequent occurrence of spelling and grammatical errors and data sparseness and lack of context due to the short length of the messages. The wide variety of topics discussed by the users of these services, may also present some challenges. In the case of event detection, for example, it may prove challenging to separate the mundane information (e.g. daily chatter) from the real-world events of interest [5].

This diversity of possible applications along with the many challenges just mentioned contribute to the relevance of this field of research, which is also the topic of this work.

1.2 Goals

The main goal of this work is to implement a fully functional system to detect newsworthy events using tweets and then test this implementation. By newsworthy event, it is meant any real-world event of sufficient interest to be reported in the Media, for example in newspapers, television or online news. The focus of the work is on the portuguese language, as opposed to the english language, more commonly used in research work, therefore the dataset used should comprise tweets written mostly in this language. Twitter was chosen in particular as it is used in several similar proposed systems, some of which are presented in Section 2.4 when discussing the related work.

To achieve this goal a similar system already proposed in the literature, specifically the Twevent system presented in [10], is used as the base of the implementation. This serves not only as a sound starting point but also as a reference that can be used to compare the results obtained and assess the validity of the implementation. This specific system was chosen for the following reasons:

- the paper referred is cited by a fair amount of other works in the area (49 citations in ACM Digital Library at the time of the writing of this document);
- the authors themselves have several other published work also in this area and are reasonably cited;
- the system proposed in the paper is used as the base of other proposed systems in the literature such as in [11];
- the system is well described and most of its implementation is also formalized, which means it can more easily be reproduced and implemented.

Finally, the use of this specific base system adds two more goals to this work. The first of these goals intends to empirically validate the proposal of introducing Wikipedia as an additional factor in the computation of the weighting scheme used to rank the segments, relatively to the original formulation. This change is proposed in order to try to boost segments potentially more newsworthy to the top of the ranking, mostly dominated by more commonly used and less informative segments. The second goal stems from the fact that the focus of this work is on the portuguese language. This then presents the opportunity to compare the results obtained, with those of similar implementations targeting different languages, such as the system proposed in [11], targeting the english language.

It should be also noted that the implementation of the system proposed in this work, while following some of the guidelines and the general formulation of the system used as its base, does not use any pre-existing code from that same system.

1.3 Motivation

The main motivation behind this work was to delve in a research area with many possible applications, some of which with significant social impact, such as the detection of natural disasters and diseases, but also very challenging due to the nature of the data used and to some of the specificities regarding the design and usage of the sources of these data, that is, the Social Media services themselves. An additional motivation was the possible usage of Machine Learning techniques in the resolution of this problem, as this field of research if also of interest to the author. This can serve also as a complement to the curriculum of the Masters in Informatics Engineering as Machine Learning was not a relevant topic in the curriculum and was only briefly introduced and explored.

Specifically regarding the system used as the base of the implementation, two additional problems served as further motivation, specifically the implementation of a segmentation algorithm using a dynamic programing approach and the implementation of a variant of the Jarvis-Patrick clustering algorithm, both of which are used and referred to in the original paper but not formalized in terms of their implementation.

1.4 Contributions

The main contribution of this work is the implementation of a system to perform the detection of newsworthy events in Twitter along with its supporting infrastructure, where the following points are highlighted:

- the proposal and implementation of a tweet segmentation algorithm using a dynamic programming approach in order to allow for an efficient and scalable solution to the tweet segmentation problem;
- the proposal of a revised formulation of the weighting scheme used to rank and retain the top *K* segments selected to be further processed. In this regard Wikipedia is proposed as an additional factor in the computation of this weighting scheme. This is done in order to boost potentially more newsworthy segments up in the rank and counter the dominance of more common use ones, due to their greater user support. The validity of the proposed revision is checked empirically.
- the implementation of a variant of the Jarvis-Patrick clustering algorithm;

- the tuning, training and testing of an SVM model to be used in the filtering step. For this purpose a subset of the statistical and social features proposed in [11] is used. This SVM model is used with the expectation that this model can better capture the distinctive features that relate a candidate event to a real-world newsworthy event and therefore obtain better results in terms of accuracy, while at the same time avoiding the time consuming process of tuning a user defined threshold value, as proposed in the base system [10].
- the implementation of the supporting infrastructure to manage the precomputed values supporting a language other than english;
- the testing of the system, using tweets mostly written in the portuguese language and the comparison of the results obtained with regard to other similar implementations;
- the discussion of the properties of the system and its applicability to the task proposed.

It should also be noted that initially this work was to be implemented using a cognitive system, namely IBM Watson [12] by resorting to the cognitive APIs available at the Bluemix service, IBM's cloud development platform [13]. The conditions imposed on the free utilization of these services however proved to be too restrictive and this option had to be abandoned. A paper concerning the prospect of the utilization of cognitive systems to perform access control, of which the author of this work is also a co-author was nevertheless published in [14].

Finally this work also originated the submission of a paper titled "*Discovery of Newsworthy Events in Twitter*" to the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018) [15] of which the author is also a co-author, describing the details regarding the implementation of the system proposed in this work, as well as the results obtained. At the time of this writing this paper is waiting review and acceptance.

1.5 Outline

The reminder of this document is structured as follows:

- **Chapter 2** presents the state of the art. The theoretical foundations as well as the technical background relevant to this work are presented. The related work is also reviewed;
- **Chapter 3** presents the conceptual architecture of the system to be implemented in this work. The requirements of the system, its architecture and the workflow of the data are some of the topics discussed;
- **Chapter 4** details the implementation of the system. Special emphasis is given to the implementation of the main components as well as to the setup process of the infrastructures created to manage both the dataset used, as well as the precomputed values required;

- **Chapter 5** presents the analysis conducted on the dataset used. This analysis was conducted as a pre-step in order to gain a deeper insight about the data and make more informed decisions during the implementation phase;
- **Chapter 6** presents the results obtained during the testing phase of the implemented system as well as other performance information found relevant. These results are also compared to those obtained by similar implementations;
- **Chapter 7** concludes the document, by presenting the main difficulties encountered, future work and the conclusion.

Summary

This chapter presented the overall outline of this work. The discussion presented the several goals to be achieved, as well as the main motivations that led to the choice of this theme. The contributions resulting from this work were also presented.

2 State of the Art

This chapter presents the conceptual and technological background knowledge considered most relevant in the scope of this work and its purposes, giving special emphasis to the techniques and algorithms used. The rationale behind the choices of these techniques and technologies is however left out of the presentation and shall be addressed in full detail, later on in this document, during the discussion of the implementation of the system. Also, some of the concepts presented, are given their formal definition, and in some cases this is only done for the purpose of completeness of the discussion, as the full understanding of these mathematical formulations is not required in order to understand the remainder of the presentation. This will be pointed out when appropriate.

Given this, the chapter starts by discussing Social Media in terms of its growth, relevance, relevant research topics, with a specific focus on event detection and some of the challenges it presents in terms of data management and analysis. An overview of Machine Leaning is discussed next, giving special emphasis to the techniques and algorithms used in this work. Ensuing this, other relevant concepts also used, are presented for better contextualization of some of the implementation details to be discussed later on. The discussion of the state of the art concerning event detection research in Social Media is discussed next. A review of the technologies required to build the proposed framework concludes the chapter.

2.1 Social Media

This section presents some relevant aspects related to Social Media, such as its growth and relevance, both in modern society as well as in research. Twitter is presented as an example of a Social Media service. Event detection, a research field associated with Social Media, is then discussed as well as some of the main challenges encountered when doing research related to Social Media data.

2.1.1 Growth and Relevance

The term Social Media is used to refer to the collective of web-based tools and technologies whose purpose is to allow and facilitate the creation, exchange and sharing of User Generated Content (UGC) [16]. Social Media can be classified and divided into several different types. In [16] six types of Social Media are identified, namely: **collaborative projects**, **blogs**, **content communities**, **social networking sites**, **virtual game worlds** and **virtual social worlds** and classified according to four parameters: **social presence**, **media richness**, **self-presentation** and **self-disclosure**. Microblogging a form of blogging [4], is addressed in more detail in the next subsection.

Social Media has become a popular medium of communication amongst users, as shown by its continuous growth in terms of its user base [17]. As of the second semester of 2017, regarding the number of monthly active users worldwide, for example, the microblogging service Twitter is estimated to have 328 million [18], the social networking service Facebook [19] around 2 billion [20] and the photo sharing network Instagram [21] reaching 800 million [22]. This in turn contributes to the relevance of Social Media. In fact data generated from Social Media sources can be of great value and interest to both business enterprises and researchers, as these data reflect aspects of real-world societies and are generally publicly accessible thru web crawlers and specialized Application Programming Interfaces (API) provided by the Social Media platforms themselves [1].

Proof of this value and relevance is the wide range of data analysis applications where Social Media data has been used, such as in **sentiment analysis** and **opinion mining** [23], **trend detection** [24], **user influence** [25], **knowledge cascades** and **information propagation** [26], **community dynamics** [3], **disease tracking and prediction** in healthcare [27] and **event detection** [10] also discussed in more detail in a later subsection.

2.1.2 Microblogging and Twitter

Microblogging is a form of blogging. In the case of microblogging however user content is limited in size. Several formats of content can be used though such as short text messages, links, images and video [5]. One possible positive side effect of this limitation is that content can potentially be easier to consume, due to its greater conciseness, and be spread to other users more rapidly (e.g. real-time information can be disseminated by any user to the rest of the world as events unfold). This immediacy is also one of the distinctive characteristics that contributed to the popularity of these services [5].

Twitter is a well-known example of a microblogging service that allows users to post short status updates, also called tweets, limited to 140 characters in length about any topic of their choice. For this purpose various communication services can be used, such as emails, mobile

phones and web interfaces [5]. This in turn alludes to another characteristic, not specific to Twitter, but common amongst microblogging services, which is their portability [5].

Twitter also provides a well-defined markup vocabulary. In Twitter messages, for example, placing the @ symbol before a user name, also called a *handle*, creates either a *mention* or a *reply* to that users' account depending on whether this handle occurs at the beginning of the message (reply) or anywhere else in the message (mention). Keywords preceded by the # sign, called *hashtags* are used to categorize topics and create and follow threads of discussion. Finally the *RT* prefix is used to signal a given tweet as being a retweet. The retweet action is supported by Twitter as an easy way for users to forward someone else's tweets to their followers while giving credit to the original user and without needing his or her permission to do so [5].

In terms of social relationships in Twitter these are asymmetric as a user can follow any other user without requiring approval or a reciprocal connection from the followed users. Twitter does not impose any limit on the number of followers to a user account. Also, by default posted messages are available publicly to anyone, although users have some privacy options available to setup their accounts, so that only their followers receive their posts and to decide upon those who can follow them [5].

Finally the fact that Twitter is updated worldwide at a very fast rate by a continuous flow of status updates, containing diverse amounts of information such as daily-life occurrences or the latest local and international news and events, that these tweets also include additional metadata such as time, source, and location, and that these data are publicly available and can be easily collected using the API provided by Twitter, all contribute to the potential of Twitter as a source of information and object of research [5]. These are also some of the reasons why Twitter was chosen as the data source of this work. This topic is covered in more detail later on in this document, when discussing the conceptual architecture of the system.

2.1.3 Event Detection in Social Media

Event detection in Social Media is a research field with many possible applications, such as for example in the detection of emergent newsworthy topics for news professionals [8] and in the detection of natural disasters and epidemic breakouts [6]. This diverse range of applications however may introduce some inconsistency concerning the definition of event detection itself. This issue is addressed in [1], where a unifying definition of event detection in an Online Social Network (OSN) is proposed.

According to this definition, event detection consists on the task of identifying a set of real-world events, given a stream of actions in the OSN, and provide some information such as: 1) the textual representation of the event; 2) a set of actions related to the event; 3) a temporal definition

of the set of actions; 4) a location correlated to the event; 5) the actors involved. An action in turn is defined as either: 1) a post of new content (e.g. a new tweet); 2) an interaction with another profile (e.g. a new follower); 3) an interaction with another user's content (e.g. a retweet).

The definition of event however differs from author to author. In [28] an event is defined as a set of related words showing an increase in usage, [29] defines newsworthy events as *"temporal bursts of closely related posts",* [30] defines an event as *"a significant thing that happens at some specific time and place"* and [31] defines an event as "*a real-world occurrence e with (1) an associated time period* T_e *and (2) a time-ordered stream of Twitter messages* M_e*, of substantial volume, discussing the occurrence and published during time* T_e*"*.

To address this issue [1] also proposes a unifying definition and defines event, in the context of Online Social Networks, as a *"(significant) event e is something that causes (a large number of) actions in the OSN*". It is also proposed to measure the importance of an event by the amount of the actions it generates. For the purposes of this work, the unifying definitions proposed for both the concept of event detection and event are used, in an effort to follow a standardized definition of these concepts. This topic is covered in more detail later on in this document, when discussing the conceptual architecture of the system.

In terms of the techniques and methods proposed for the purposes of event detection, specifically in Twitter, the choice is diverse. A taxonomy is presented in [5], whereby these techniques are categorized as: **unspecified event detection** and **specified event detection** depending on the type of event, **New Event Detection** (NED) and **Retrospective Event Detection** (RED) according to the detection task and **supervised** and **unsupervised** techniques depending on the detection method.

Unspecified event detection techniques rely on temporal signals found in the Twitter stream, such as bursts or trends to detect events, as no prior information is available about the events themselves. *Specified event detection* techniques on the other hand rely on specific information known about the events such as venue and time.

NED techniques are used to discover new events in near real time by continuously monitoring the Twitter stream for the occurrence of signals indicating evidence of possible events. RED techniques on the other hand focus on event retrieval from historical data.

Finally *unsupervised detection* techniques are often used to perform unspecified event detection using clustering approaches, such as K-Means and K-Medoids [32]. On the other hand *supervised detection* techniques, such as Naïve Bayes [31], are mainly used when performing specified event detection. Both of these techniques rely on methods drawn from other fields, such as Machine Learning (ML), Natural Language Processing (NLP) and Information Retrieval [5].

2.1.4 Challenges of Social Media Data

This subsection presents some of the challenges encountered when performing research related to Social Media data and specifically in the task of event detection.

Big Data

The term Big Data is usually employed when referring to data that due to its characteristics requires the deployment of new hardware and software architectures, techniques and algorithms in order to be properly managed [33]. Some of these characteristics are summarized in what is referred to as the 7 V's of Big Data [34]:

- **volume**, referring to the vast amounts of data being generated every second (the data itself however may be small);
- **velocity**, referring to the speed at which data is generated and must be processed;
- **variety**, referring to the plethora of different types (e.g. text, image, video), structures (structured, unstructured or semi-structured) and formats (e.g. Javascript Object Notation (JSON) [35], Extensible Markup Language (XML) [36]) in which data can present itself;
- **veracity**, referring to the truthfulness or how certain one is about the data;
- **validity**, concerning the correctness and accuracy of data with respect to its intended usage;
- **volatility**, referring to the policies associated with data retention and its implications in storage cost and;
- **value**, referring to the relationship between the value extracted from data and the costs associated with its ownership and management.

In [37] Social data is considered as a type of Big Data and therefore any task dealing with this data, such as in the case of event detection in Social Media, may potentially have to deal with some of the challenges inherit to a few or all of the characteristics just mentioned.

Specifically and for the purposes of this work, while *variety* and *veracity* may not be as relevant, because: 1) mainly the textual content of the tweets is taken into account and the format of the data is the one specific to the targeted platform, Twitter in this case (largely reducing variety); 2) these data can be obtained from the API provided by the platform itself (greatly asserting the veracity of the data), *validity* may have a considerable impact in the results obtained, due to content polluters such as spammers for example, attracted by the popularity of these Social Media services [38].

Concerning the last Vs, *volume*, *velocity* and *volatility*, these are also of concern in the scope of this work. As an example of this, it is estimated that on average, around 6,000 tweets are

created every second in Twitter [39], the data source used. This greatly contributes to the concerns related to *volume* and *velocity* (vast amounts of data being generated at high speed: 350,000 tweets sent per minute, 500 million tweets sent per day [39]) and also *volatility* (manage the storage of these data). The system should therefore be as performant as possible in computational and storage terms in order to maximize the *value*.

Noisy Data

Social Media data can be sparse and very noisy. Twitter for example poses some challenges inherit to its design and usage [5]. On one hand tweets are very short due to the limitation imposed in their size, on the other hand, its informal nature leads to the frequent use and occurrence of informal and abbreviated words, spelling and grammatical errors, improper syntactic structure and mixed languages. This data sparsity and diversity of vocabulary render traditional natural language processing techniques less suitable for this kind of data.

The wide variety of different topics, potentially present in this data (e.g. daily chatter, conversations, reporting news, spam, etc), may also prove to be challenging. In the case of event detection for example, this means that the event detection system must be able to properly separate the events of interest from the non-interesting ones [5].

2.2 Machine Learning

This section presents an overview of Machine Learning, focused mainly on the techniques, concepts and algorithms used in this work. The rationale behind these choices, as already stated, is presented in more detail later on in this document when discussing the implementation of the system. The section starts by presenting a definition of Machine Learning as well as some of its fields of application. Supervised learning and unsupervised learning, two types of Machine Learning, are then presented and discussed in more detail.

2.2.1 Definition and Applications

Machine learning evolved as a subfield of artificial intelligence and has as its main goal the development of algorithms that can learn from data to automatically extract knowledge from it [40]. More specifically [41], a Machine Learning algorithm wraps a model, defined by a set of parameters, and the learning step consists in the optimization of these parameters using training data or past experience. This model can be used to make predictions in the future, that is, generalize to yet unseen data, in which case the model is said to be **predictive**, or be used to gain knowledge and insight from data, in which case the model is said to be **descriptive**. The model can also be

both predictive and descriptive. Several techniques from the field of statistics are employed in building these models.

Machine Learning has many applications both in the industry and research worlds such as [41]: **basket analysis** (i.e. finding associations between products bought by costumers) in retail; **credit scoring** (i.e. calculating the risk given the amount of credit and the information about the customer) and **fraud detection** in banking; **optical character recognition** (i.e. recognizing character codes from their images) in pattern recognition; **spam filtering** and **machine translation** in NLP; **speech recognition** and **biometrics** (i.e. recognition or authentication of people using their physiological and or behavioral characteristics). Machine Learning techniques are also used to perform **event detection** in Social Media as already stated during the presentation of this same topic earlier on in subsection 2.1.3.

In terms of its subfields, Machine Learning is generally divided into three categories [40]: **supervised learning**, **unsupervised learning** and **reinforcement learning**. A fourth category, namely **semi-supervised learning**, halfway between supervised and unsupervised learning is also sometimes referred [42]. Supervised as well as unsupervised learning are discussed next.

2.2.2 Supervised Learning

This subsection presents supervised learning. The subsection starts by presenting an overview of the goals as well as some of the applications of this type of Machine Learning. Some commonly used techniques in order to tune and assess the performance of the models used to perform classification tasks are also presented. Support Vector Machine, an algorithm used to perform several supervised learning tasks, is presented at the end of the subsection. Although some formulation details of this algorithm are presented, this presentation is not intended to be thorough and is included for the purpose of completeness of the discussion, as the understanding of these formulations is not required in order to understand the remainder of the document.

The goal of supervised learning is to find a mapping from x to y given a training set composed of labeled pairs (x_i, y_i) [42]. In this case $x_i = (x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(d)})$ denotes a sample or observation composed of d features (i.e. attributes), with $x_i^{(1)}$ standing for the first feature, $x_i^{(2)}$ for the second and so on and y_i denotes the label of x_i. The term *labeled data* (or pairs) refers to the fact that each sample is annotated with a label (this process is usually manual). That is, the desired output for each sample is known in advance and is provided to the algorithm in the training phase. This can be seen as a form of supervision hence the name supervised learning. This mapping can then be used to make predictions about yet unseen data.

Supervised learning is generally divided into two subcategories [40]: **classification** and **regression**. When y takes values in a finite set the task is called *classification*, when y takes

continuous values the task is called *regression*. Binary classification is a typical example of classification where y takes only two possible values usually 1 or 0. As an example of each of these supervised tasks, trying to classify an email as spam or not spam where y = (*not spam, spam*) is an example of a classification task, while trying to predict the cost of a house given its attributes where $y \in \mathbb{R}$ is an example of regression. Decision trees, Naïve Bayes, K-Nearest Neighbors and Support Vector Machine (SVM) are some of the algorithms used in supervised learning [43]. In the scope of this work only classification is used.

Several techniques exist (both for classification as well as for regression) to help tune the parameters of these models, as well as to obtain a more reliable estimate of the model's generalization error (i.e. how well the model performs on unseen data). **Cross Validation** (generalization error estimation) and **Grid-Search** (model parameter tuning) are two examples of such techniques [44]. In *cross validation* the training set is divided into v subsets of equal size also called v-folds. The classifier is then trained with the data of $v - 1$ of these folds and tested on the remaining fold. This procedure is done sequentially so that all folds are tested individually in turn against the remaining $v - 1$ folds. Cross validation can help prevent the overfitting problem (i.e. the classifier performs well during training but does not generalize well to new unseen data).

Grid-Search can be used to tune the hyper-parameters of the classifier by exhaustively testing different combinations to find the one that yields the best performance in terms of its prediction accuracy. In the case of SVM the hyper-parameters to tune are the *kernel*, C and γ. This technique however can be computationally expensive. Cross validation and grid-search can also be used together.

The performance of a model can be measured by several performance metrics. In the case of classification models, this performance can be measured using for example the **precision** and **recall** metrics [40]. Considering a binary classification problem, that is, where only two classes or labels exist, and denoting these labels as the positive p and negative n classes respectively, these measures can be defined as follows [43]:

- *precision* measures the fraction of instances that actually belong to the positive class from those the classifier predicted as belonging to that class. The formula for precision is presented in Equation 1, where *tp* denotes the true positives (i.e. instances correctly classified as belonging to the positive class), and *fp* the false positives (i.e. instances incorrectly classified as belonging to the positive class).
- *recall* measures the fraction of instances belonging to the positive class that the classifier was able to correctly classify. The formula for recall is presented in Equation 2 where *fn* denotes the false negatives (i.e instances incorrectly classified as belonging to the negative class).

A **confusion matrix** can also be used to visualize the predictions of a classifier in terms of these true positive, true negative, false positive and false negative counts [40].

$$precision = \frac{tp}{tp + fp} \quad (1)$$

$$recall = \frac{tp}{tp + fn} \quad (2)$$

Support Vector Machine

Support Vector Machine is a supervised learning algorithm that can be used for both the tasks of classification and regression [41]. More formally, given a set *X* of training data of size *l*, where each sample or observation in *X* is represented as an instance-label pair of the form (x_i, y_i), with $x_i \in \mathbb{R}^n$, $y_i \in \{1, -1\}$ and *n* denoting the number of attributes or features of the sample.

The training vectors x_i are first mapped into a higher dimensional space via the use of a function ϕ. SVM then tries to find the linear hyperplane (i.e. the decision boundary) with the maximum margin that separates the training samples into their correct labels or classes in this higher dimensional space [44]. In this context, margin is defined as the distance from the hyperplane to the instances closest to it, and the goal is to maximize this margin in order to improve the generalization of the model. All training instances lying on this margin are selected as the support vectors (i.e. the training instances defining the margins for both sides of the hyperplane) [41].

The objective function of SVM is presented in Equation 3, with *C* denoting the penalty parameter of the error term and ξ the vector of slack variables that store the deviation from the margin. This definition is given only for completeness and its understanding is not required for the purposes of this work. More details concerning the mathematical foundations of SVM and also this objective function can be found in [41].

$$\min_{w,b,\xi} \ \frac{1}{2} w^T * w + C \sum_{i=1}^{l} \xi_i \quad (3)$$

$$subject\ to \quad y_i * (w^T * \phi(x_i) + b) \geq 1 - \xi_i, \xi_i \geq 0$$

SVM can be further adapted to various other applications such as in bioinformatics and language processing via the use of kernel functions [41]. Kernel functions are used in order to simplify computation (i.e. kernels are applied in the original space of the instances with no need to map the instances into a higher dimensional space first). Another aspect concerning kernels as to do

with *kernel engineering*, meaning that any prior knowledge about the application can be provided to the learning algorithm by using appropriately defined kernels [41].

Some of the advantages of SVM are its versatility as already discussed, by allowing the use of different kernel functions, including custom implementations of these kernels and its memory efficiency due to the use of only a subset of the training points in the decision function, the so called support vectors, also alluded to earlier. SVM is also effective in high dimensional spaces and can still be effective even in some situations where the number of dimensions (i.e. attributes or features of the samples) is greater than the number of training samples [45].

2.2.3 Unsupervised Learning

This subsection presents unsupervised learning. The subsection starts by presenting an overview of the goals of this type of Machine Learning as well as some of its applications. The Jarvis-Patrick clustering algorithm, a non-iterative clustering technique used to perform several unsupervised learning tasks, is presented at the end of the subsection.

The goal of unsupervised learning is to find the structure of data [42]. In this case however data is not labeled. More formally given a dataset X composed of m instances of x_i, with $X = (x_1, x_2, \ldots, x_m)$ the goal of unsupervised learning is to find regularities in X, that is, its unknown structure. Two commonly used techniques of unsupervised learning are **clustering** and **dimensionality reduction** [40]. *Clustering* allows data to be organized into meaningful groupings (i.e. clusters) without any prior knowledge about group memberships. A cluster defines a group of objects that share a certain degree of similarity that makes them more similar amongst themselves and also more dissimilar to objects belonging to other clusters.

Dimensionality reduction is another unsupervised technique usually used to remove noise from data, reduce the dimensionality of high-dimensional data and for data visualization [40]. As an example of high-dimensional data, consider x_i with d features where d takes a high value such as 50,000. Each of these features in turn can be seen as a dimension. The dimensional space for each sample x_i in this case is therefore composed of 50,000 dimensions. This can pose a problem both in terms of computational cost and as an important factor of degradation of the predictive performance of some algorithms, the so called curse of dimensionality [40].

The goal of dimensionality reduction is therefore to project this high-dimensional space with d dimensions into a much smaller dimensional space k with $k \ll d$ while still retaining most of the relevant information contained in the data. K-Means is an example of a commonly used clustering algorithm [43]. Principal Component Analysis (PCA) is a common technique used for dimensionality reduction [40]. In the scope of this work only clustering is used.

Jarvis-Patrick Clustering Algorithm

The Jarvis-Patrick clustering algorithm is a clustering method based on similarity by sharing of near neighbors [46]. More formally let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of n points and let x_i and x_j be any two points belonging to S. Under these assumptions x_i and x_j are similar to the extent that: 1) x_i and x_j belong to each other's neighborhood; 2) x_i and x_j share the same l neighbors out of their k nearest neighbors list (i.e. have at least l neighbors in common).

Given this definition it can be easily seen that the algorithm is parameterized by two variables or parameters: k denotes the number of nearest neighbors to examine for each point and l denotes the minimum required number of common neighbors between two points regarded as sufficient for the purposes of grouping points together. To compute the neighbors for each point any suitable measure can be used.

Jarvis-Patrick can be used when working with non-globular clusters, when a deterministic clustering result is desired or when performance is an issue since the algorithm is non-iterative and the raw data can be completely discarded once the k neighbors list for each point has been computed typically with $k \ll n$ [46].

2.3 Other Relevant Concepts

This section presents a miscellaneous of relevant concepts that are used in this work, in order to provide a better contextualization of some implementation details discussed later on in this document. Some of the formulations presented, namely for both *term frequency-inverse document frequency* (TF-IDF) and *cosine similarity*, serve mainly for the purposes of completeness of the discussion, and the understanding of the underlying mathematical concepts is not required. In the case of the *symmetrical conditional probability*, the formulation is fully detailed as this same formulation is used in this work.

2.3.1 TF-IDF

The term TF-IDF is used to denote a class of term weighting schemes often used in the field of Information Retrieval. TF and IDF stand for respectively, *term frequency* and *inverse document frequency*. While TF is a measure of locality and denotes the frequency of the term in the document (i.e. how many times it occurs in the document), IDF is a global measure and stems from the intuition that terms occurring in many documents are not good discriminators and should therefore be given less weight then terms that occur in few documents (potentially more informative) [47].

Using this weighting scheme, a document can then be transformed into a vector in a vector space, by computing the weight of each of its terms or words (each term representing a dimension).

Similarity between documents in this vector space can then be computed using vector arithmetic [48].

A common form of IDF is presented in Equation 4, where *N* stands for the total number of documents in the dataset and n_i denotes the number of documents in the dataset where the term t_i occurs [47]. TF and IDF measures have several variants that can be combined together differently. Mnemonics are usually used to represent a specific combination of these variations. This system of mnemonics is sometimes also called SMART notation [49].

$$idf(t_i) = \log\frac{N}{n_i} \quad (4)$$

2.3.2 Cosine Similarity

A standard way to quantify the similarity between two documents is to compute the cosine similarity of their vector representations [48]. For this purpose, a vector is derived from each document. This vector is composed of several components, as many as the number of terms in the dictionary (i.e. one component for each term). These components are usually computed using the TF-IDF scheme. The set of documents may then be viewed as a set of vectors in vector space in which each term is represented by an axis. The formula for the cosine similarity is presented in Equation 5, where d_1 and d_2 denote the vector representation of their respective documents and *n* stands for the number of components in the vector.

$$sim(d_1, d_2) = \frac{\sum_{i=1}^{n} d_{1i} * d_{2i}}{\sqrt{\sum_{i=1}^{n} d_{1i}^2} * \sqrt{\sum_{i=1}^{n} d_{2i}^2}} \quad (5)$$

2.3.3 Collocations, Associations and Co-occurrences

A *collocation* is defined in [50] as '*an expression consisting of two or more words that correspond to some conventional way of saying things*'. Common expressions in english such as *strong tea*, *to make up* and *the rich and powerful* are examples of collocations. Three criteria are typically used to help identify collocations [50]: **non-compositionality**, **non-substitutability** and **non-modifiability**.

Non-compositionality refers to the fact that the meaning of a collocation is not the same as the composition of the meanings of its parts. In fact its meaning can be completely different. The idiom *kick the bucket* that means *to die* is such an example.

Non-substitutability refers to the fact that words in collocations cannot be substituted by other words with similar meaning, that is, synonyms, as in *fast food* and *quick food*. Although the words *fast* and *quick* are synonyms in this case, the expression *quick food* is not used.

Non-modifiability refers to the fact that most collocations cannot be modified with additional lexical material or by applying grammatical transformations. For example the idiom '*Cat got your tongue?*' cannot be modified to '*Has the big black fat ugly cat got you tongue?*'.

It should be noted however that proper nouns (i.e. a name used for an individual person, place, or organization) and terminological expressions (i.e. expressions referring to objects and concepts in technical domains), although not fully abiding by the above criteria (e.g. terminological expressions are usually highly compositional as in *hydraulic oil filter*) are usually considered as collocations in some fields of research [50].

Collocations are important for many NLP applications such as machine translation, word sense disambiguation, language generation, and information retrieval [51]. Collocations are also of interest to the field of event detection and this work because they can be thought of as semantic units, which in general can be much more informative than any of its parts taken individually (e.g. the expression *white wine* is much more informative than just *wine* or *white* individually).

The development of methods for automatic collocation extraction has therefore become a topic of research. Over the years several such methods have been proposed in the literature. An overview of the most widely used is presented in [50] while [51] presents a compendium of 84 of these methods as well as an empirical evaluation of their performances. As a side note, it should be pointed that most of these methods are based on the verification of the properties that define a collocation (the most widely tested property of collocations is non-compositionality).

These properties are formally defined by mathematical formulas called *association measures* (e.g. Pointwise Mutual Information). These association measures in turn compute an *association score* that can be used to decide if a candidate collocation should be considered as a true collocation or not. A threshold can be used for this purpose. The values used for these thresholds are however not tabulated and depend on the application [51].

The terms *association* and *co-occurrence* are usually used to refer to the fact that some words are likely to occur in the same context. These words need not occur in any particular order nor be grammatically bounded. The words *nurse* | *doctor* and *plane* | *airport* are such examples. The notion of collocation is sometimes generalized to also include these cases [50].

2.3.4 Symmetric Conditional Probability

Symmetric Conditional Probability (SCP) is proposed in [52] as a new association measure to extract lexical units, that is, two or more consecutive words or *n*-grams (an n-gram refers to a contiguous sequence of one or more words in a text) with $n \geq 2$. These so called multi-word units (MWUs) include collocations but also other frequent expressions encountered in written text such as those produced by compound nouns (e.g. Portuguese minister of foreign affairs), compound

verbs (e.g. to come into force) and compound conjunctions (e.g. in order to). In this regard, SCP is presented as a function that measures the 'glue' sticking together every two adjacent words within a candidate n-gram. Before formally introducing SCP however, two concepts should first be introduced: **n-grams' dispersion points** and the **pseudo-bigram transformation**.

The definition of *n-grams' dispersion points* states that every 2-gram has one and only one dispersion point located between the positions of its words, that is, between the first and the second word. This definition can then be generalized to any *n*-gram with $n \geq 2$, by saying that every such *n*-gram has exactly $n - 1$ dispersion points, the first dispersion point located after the first word, the second dispersion point located after the second word and so on until the $n - 1$ dispersion point located after the $n - 1^{th}$ word (a 2-gram would have $n - 1$ dispersion points, that is 1 dispersion point. A 3-gram would have $n - 1$ dispersion points, that is 2 dispersion points and so on).

The definition of *pseudo-bigram transformation* states that although every *n*-gram has $n - 1$ dispersion points the *n*-gram may be regarded as having only one dispersion point located between the left $w_1 \ldots w_p$ and the right $w_{p+1} \ldots w_n$ parts of the *n*-gram, with $1 \leq p \leq n - 1$. This way any n-gram despite its size (i.e. the value of *n*) can be seen as a pseudo-bigram. This allows for the comparison of the 'glue' values computed for different size n-grams as well as to perform this comparison as the size of the n-gram changes. This information can then be used to select the MWUs from the candidate n-grams.

With these definitions in place SCP can now be formally defined. Given a bigram *xy* (i.e. composed by words *x* and *y*) SCP measures the association between *x* and *y* by taking the conditional probabilities of each one given the other, more specifically by taking the product of these two conditional probabilities.

Formally, SCP is defined by Equation 6 (the definition of conditional probability is used, $P(A|B) = P(A \cap B)/P(B)$), where *SCP(x,y)* stands for the Symmetric Conditional Probability of the bigram composed of words *x* and *y*, *P(x|y)* stands for the conditional probability that *x* occurs given that *y* has occurred (conversely *P(y|x)* is the conditional probability that *y* occurs given that *x* has occurred), *P(x,y)* stands for the joint probability of *x* and *y* occurring together and *P(x)* and *P(y)* stand for the probability of *x* and *y* respectively.

$$SCP(x,y) = P(x\,|y) * P(y|x) = \frac{P(x,y)}{P(y)} * \frac{P(y,x)}{P(x)} = \frac{P(x,y)^2}{P(y) * P(x)} \quad (6)$$

This definition can be further generalized to any generic *n*-gram with $n \geq 2$, by transforming the n-gram into a pseudo-bigram with a dispersion point located at position *p*, with $1 \leq p \leq n - 1$, as previously stated in the definition. Equation 7 presents the generalized formula for the SCP measure, where $P(w_1 \ldots w_k)$ stands for the joint probability of all words occurring between and including the word w_1 and the word w_k.

In the case of *n*-grams with $n \geq 3$, the location of this dispersion point however is not unique and will certainly affect the values computed by the SCP. To solve this problem, fair dispersion point normalization is introduced as depicted in Equation 8. This technique consists in taking the arithmetic average of the products of all the *n – 1* dispersion points along the n-gram.

$$SCP\left(\left(w_1 \dots w_p\right), \left(w_{p+1} \dots w_n\right)\right) = \frac{P(w_1 \dots w_n)^2}{P\left(w_1 \dots w_p\right) * P(w_{p+1} \dots w_n)} \quad (7)$$

$$SCP(w_1 \dots w_n) = \frac{P(w_1 \dots w_n)^2}{\frac{1}{n-1} * \sum_{i=1}^{i=n-1} P(w_1 \dots w_i) * P(w_{i+1} \dots w_n)} \quad (8)$$

2.4 Related Work

This section presents some of the most relevant work on event detection in Social Media proposed in the literature. The discussion also aims to address the main differences between these systems and the system to be implemented in this work.

TvPulse [53] is a project developed at Instituto de Telecomunicações de Aveiro (IT) in partnership with PT Inovação [54] (now Altice Labs) that aims to detect TV highlights using Twitter and publicly available Electronic Programming Guides (EPGs). For this purpose tweets created in Portugal and therefore mostly written in Portuguese, were collected (since the focus of the research is on the Portuguese community) as well as TV programs. The collected tweets and TV programs were then processed in order to compute several features associated to them. Distributional profiles for the Portuguese language were also computed from these tweets in order to build semantic profiles. These semantic profiles were then used to identify the most representative tweets as highlights of a TV program.

Hotstream [55] is an application proposed to collet, group, rank and track breaking news in Twitter. For this purpose tweets are filtered by hashtags (e.g. #breakingnews) or keywords (e.g. breaking news) often used by users to annotate breaking news. The similarity between the tweets is then computed using TF-IDF along with a boost factor obtained via the use of a Named Entity Recognizer (NER). Tweets are then grouped together according to this similarity measure.

In [56] a method is proposed to automatically detect events involving known entities from Twitter. A snapshot, that is, a set of tweets created over a period of time referring the target entity, is collected. The task to decide whether this snapshot describes a central event involving the target entity or not, is then formulated as a supervised machine learning problem and solved using the Gradient Boosted Decision Trees framework.

Tedas [57] is a Twitter based event detection and analysis system that aims to detect new events, rank them according to their importance and compute both their temporal and spatial

patterns, with a special focus on the detection of Crime and Disaster related Events (CDE). To achieve this, spatial and temporal meta information is extracted from tweets and then indexed by a text search engine along with the original tweets. This index can then be used to retrieve real time CDEs or answer analytical queries.

More recently in [58] an event detection framework is proposed to detect large and related smaller scale events, with a special focus on the detection of disruptive events, that is, incidents that may jeopardize social safety or provoke social disorder. To achieve this, a Naïve Bayes classifier is first trained on a manually labeled training dataset, where tweets are labeled into two classes 'event' and 'non-event'. This classifier is then used to filter out non-event related tweets and retain only those associated to events, referred to as large-scale events.

An online clustering algorithm is then used in order to cluster these tweets according to their similarity in order to obtain the smaller-scale events. The cosine similarity together with a set of temporal, spatial, and textual features is used to compute this similarity. The topics being discussed in each of these clusters are then summarized and represented by their most representative posts or their top terms, in order to make them easily interpretable by human decision makers, such as for example for law enforcement purposes. Temporal Term Frequency–Inverse Document Frequency (TF-IDF), a novel approach to TF-IDF which does not require prior knowledge of the entire dataset, is proposed in order to compute a summary of these top terms.

The systems just presented aim to detect specific types of events, or events related to specific entities or locations.

In [24] a trend detection system called TwitterMonitor, is proposed in order to identify emerging topics in Twitter in real time. Bursty keywords (i.e. words exhibiting an unusual high rate of occurrence in the tweets stream) are detected first, using an algorithm based on queue theory, and considered as an indication of the emergence of a new topic. These bursty keywords are subsequently grouped into trends based on their co-occurrences in tweets, using a greedy strategy. After a trend has been identified, it is further analyzed in order to try to compose a more accurate description of it by using the context (i.e. the other words also occurring in the same tweets) related to its bursty words. Several context extraction algorithms such as Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) are used for this purpose.

In [59] a system is proposed to tackle the problem of new event detection also known as First Story Detection (FSD) (i.e. identify the first story that discusses a particular event from a set of stories) using an algorithm based on Locality-Sensitive Hashing (LSH). Using this algorithm each tweet is assigned a novelty score based on cosine similarity. Threads of tweets (i.e. subsets of tweets that discuss the same topic) are then formed using these scores.

EDCoW (Event Detection with Clustering of Wavelet-based signals) [28] is another system proposed to detect events in Twitter streams. Wavelet analysis is used to build signals for individual words. Trivial words are then removed according to their signal auto-correlations. A modularity-based graph partitioning technique is used to cluster the remaining words into events.

This system may suffer from scalability issues as signals must be computed for each word, in order to filter the trivial ones. Furthermore, using just single words to describe the detected events, may not provide enough context about these events, making their analysis harder to be interpreted by a human.

More closely related to the work presented in this dissertation and also used as its base is Twevent [10], a segment based event detection framework for tweets. First the tweets are divided into time windows of fixed size in order to perform the event detection inside each of these time windows separately. For each of these time windows, each of its tweets is first split into non-overlapping segments, referred to as tweet segments, where a segment can be a sequence of one or more words and may or may not represent a semantic unit (e.g. the name of a person). Wikipedia is used to help improve the quality of these segments as potential semantic units.

These tweet segments are then ranked in descending order using a weighting scheme that takes into account their frequency probabilities and user support (the number of unique users that created tweets containing that segment). The top K of these segments, also referred to as event segments, are then grouped together into candidate events according to their similarity, using a variant of the Jarvis-Patrick clustering algorithm. For this, a pseudo document consisting of the tweets that refer to each event segment is computed and transformed into the TF-IDF scheme and the cosine similarity is used to compute the similarity between event segments.

The temporal feature patterns of these event segments are also taken into account in this similarity computation, by further subdividing the time window into non-overlapping sub-time windows. The resulting candidate events are then filtered in order to retain only those considered to be related to a real-world event, also referred to as real events. This is achieved by testing a ratio involving the newsworthiness scores computed for each of these candidate events, again leveraging Wikipedia, against a user specified threshold. The resulting real events are described by their most newsworthy event segments. Again Wikipedia is used to achieve this purpose.

The weighting scheme proposed by this system, in order to rank the tweet segments, can result in segments of more common use (and therefore potentially less informative) to be favored and dominate the top of the ranking, due to their greater user support. As a consequence of this, less common use segments (potentially more informative), may be further pushed down the ranking, increasing the likelihood of their removal from further processing.

FRED [11] further expands Twevent by considering three types of features representing the statistical, social and textual information related to the candidate events obtained and then using these features to train an SVM model to perform the filtering step.

2.5 Technology Review

This section presents the technologies used to build the proposed system. The section starts by presenting Python, the language chosen to code the implementation of the system. The rich ecosystem associated with Python is also presented, giving special emphasis to the modules and packages used in the implementation. Next, two database families, namely Document-Based Stores and In-Memory Databases are discussed. An example database is presented in some detail for each of these families. The databases presented are those used in this work. The rationale behind the choice of these databases is however not addressed and is discussed in detail later on, during the discussion of the implementation of the system.

2.5.1 Python

Python [60] is a cross-platform open-source programming language that can be freely used and distributed even for commercial purposes. Python is also an interpreted language (i.e. Python code is not compiled, thought highly optimized), as opposed to C or Java for example. Interpreted languages however are slower than compiled ones when dealing with numerically heavy algorithms [61].

In Python these more computationally demanding tasks can be off-loaded to extensions written in languages more efficient for these purposes such as C. Cython [62] for example, an optimizing static compiler offers such capabilities. Many Python packages also use such strategy as is the case for example with Numpy, a package for scientific computing with Python.

Python is also a very modular language and its core functionalities can easily be extended via the use of external packages and modules. PyPI [63], the Python Package Index, hosts 116,158 such packages at the time of writing and many more have been written by other third-party developers, allowing for a broad range of applications for Python, such as in Web Development, Scientific and Numeric Computing and Machine Learning.

To manage this plethora of packages and their dependencies more easily, Python also supports virtual environments, allowing for different Python installations, addressing diverse purposes, to be deployed alongside each other with no additional complexities. Also and to further alleviate the usually time consuming task of installing packages, managing dependencies and creating environments, several specialized Python distributions are available, such as Anaconda [64], a data science platform containing 150 packages pre-installed. A brief mention to the

packages used in the scope of this work is presented next, all of which are open-source and actively maintained.

Numpy

Numpy [65] is a package for scientific computing with Python. Amongst other capabilities, Numpy provides support for efficient and highly optimized multi-dimensional array and matrix structures (via integration with C/C++ extensions) as well as support for their operations as defined in linear algebra. These data structures are used by most ML algorithms.

Pandas

Pandas [66] is a Data Analysis Library for Python that provides high-performance, easy-to-use data structures and data analysis tools. Pandas is built on top of Numpy and provides a richer and higher-level layer for easy manipulation of its data structures, including support for many database-like methods such as join, merge and group by. Pandas also provides other functionalities such as support for time series, and many enhancements that make statistics tasks easier such as support for missing values in data.

Matplotlib

Matplotlib [67] is a Python 2D plotting library for data visualization capable of producing high quality figures that can be saved to disk in a variety of formats including png, pdf, ps, eps and svg. Matplotlib can generate a variety of different plots including histograms, power spectra, bar charts, error charts and scatterplots. Seaborn [68], a Python visualization library specialized in statistical data visualization is based on Matplotlib.

Scipy

Scipy [69] is a Python-based ecosystem of open-source software for mathematics, science, and engineering. Numpy, Pandas, Matplotlib, IPython and Sympy [70], for symbolic mathematics are amongst its core packages.

Jupyter

The Jupyter [71] notebook is a web application that allows documents or notebooks to be easily created and shared. These notebooks can contain live code, equations, visualizations and explanatory text and can be written in more than forty different languages including Python, R [72] and Julia [73]. Notebooks can also be easily shared via email, Dropbox and the Jupyter Notebook

Viewer [74] for example. Code written in these notebooks can produce rich output such as images and video and data can be visualized and manipulated in real time via interactive widgets.

Finally, Jupyter provides Big Data integration by leveraging Big Data tools such as Apache Spark [75]. A multi-user version of Jupyter, the JupyterHub [76], designed for companies, classrooms and research labs is also available. Jupyter is being used by companies such as Google [77], Microsoft [78] and IBM [79] and its many uses include: statistical modeling, data visualization transformation and cleaning and Machine Learning.

NetworkX

NetworkX [80] is a package for Python that offers several solutions to work with complex networks (i.e. social, biological, and infrastructure networks) including the ability to generate and visualize several types of classical and random networks, build network models, analyze the structure of networks and design new network algorithms. NetworkX offers support for various types of graphs such as digraphs (i.e. directed graphs) and multigraphs and provides a wide range of standard graph algorithms such as algorithms to compute shortest paths.

NetworkX has many fields of potential applicability such as in mathematics, physics, computer science and social science [81]. Although not specifically designed for Machine Learning tasks, NetworkX can still find applicability in this field as several Machine Learning tasks such as Graph-based Point Anomaly Detection and Graph-based Group Anomaly Detection, from the field of Social Media Anomaly Detection, rely on graphs and graph theory [82].

Scikit-Learn

Scikit-Learn [86,87] is an open-source Python module distributed under the simplified Berkeley Software Distribution (BSD) license [85] that provides state-of-the-art implementations of many well-known Machine Learning algorithms for both supervised and unsupervised learning tasks. Its strong emphasis on ease of use, performance, documentation, API consistency and minimal dependencies have contributed to encourage its use in both the academic as well as the commercial realms [84].

Scikit-Learn differs from other similar packages for Python for various reasons such as the fact that it is distributed under the BSD license, a family of permissive free software licenses that impose minimal restrictions on the use, modification and redistribution of the covered software [86], the incorporation of compiled code for efficiency, minimal dependencies, namely Numpy and Scipy and imperative programming style [84]. Scikit-Learn has also been distributed as part of major free software distributions such as Ubuntu [87] and Debian [88] and provides binary

packages for other platforms including Windows and Portable Operating System Interface (POSIX).

Finally, Scikit-Learn supports several strategies to allow computational scaling up when dealing for example with sources of Big Data such as Social Media data, that often cannot be processed using traditional approaches. Amongst these, **out of core learning**, a technique used to learn from data that cannot fit entirely into memory and **online learning**, the ability to learn incrementally from a mini-batch of instances are such examples [89]. Also its tight integration with Python and its scientific ecosystem make it easy to integrate with applications and widen its range of use outside statistical data analysis applications [84].

NLTK

The Natural Language Toolkit (NLTK) [93,94] is a free open source Python module often used in NLP research, in topics such as Named Entity Recognition (NER) and Part-of-Speech tagging (POS). It includes amongst many others, a corpora composed of several corpuses (including annotated corpora) often used in research, many lexical resources of which WordNet [92] is an example and a suite of text processing libraries for classification, stemming, lemmatization, tokenization, tagging and much more.

2.5.2 Document-Based Stores

This subsection presents Document-Based Stores. The subsection starts by introducing this family of database related products, as well as some of the more relevant concepts associated with it. An example of such a database is also presented.

Document-based databases are a family of non-relational database products, part of the NoSQL movement, that store data as structured documents typically in XML or JSON formats. XML databases are usually used as content management systems (i.e. organizing and maintaining text files in XML format such as academic papers) while JSON databases serve mostly to support web-based operational workloads of modern web-based applications [93]. For the purposes of this work only JSON databases are considered and will be the focus of discussion.

The hierarchy of storage on these databases comprises two levels: the **document** and the **collection** [93]. The *document* is the basic unit of storage. It is comprised of one or more key-value pairs and may also contain other nested documents and arrays, allowing for very complex hierarchical structures. Nested documents can be used for example, to represent master-detail relationships. While no schema is enforced on the structure of a document, its size may be capped (e.g. MongoDB limits the size of a single document to 16 Megabytes (MB) [94]). The document in

turn is treated as a whole (i.e. the database avoids splitting the document) and is usually identified by a unique primary identifier.

Documents can be indexed not only on their primary identifier but also on their other properties (i.e. the keys of the key-value pairs). A document can be thought of as the same as a row in a relational database. The fields composing the documents (i.e. the key-value pairs) are analogous to columns. A *collection* is a set of documents. These documents do not have to be of the same type. Usually however collections are created to store together documents containing some common category of information or sharing a similar structure. A collection is roughly equivalent to a table in a relational database.

The document model itself may present several advantages: the fact that all data for a given record tends to be in a single document favours data locality which drastically reduces the need for joins (i.e. a single read to the database can retrieve the entire document) improving performance and scalability, the documents are themselves self-describing and can vary in structure independently of each other (i.e. each document only needs to have the fields to which it has values assigned to), and finally, this model can also make development easier as the documents may be more aligned to the structure of objects in programming languages [94]. Couchbase [95] and MongoDB [96] are two well-known examples of this family of database products. MongoDB is presented next.

MongoDB

MongoDB is a JSON-oriented document database offering both a free open source edition (i.e. Community Server) as well as a commercial one (i.e. Enterprise Server). Amongst its wide range of products, MongoDB also offers Database as a Service solutions (i.e. MongoDB Atlas), Ops management (i.e. Ops Manager), Graphical User Interface (GUI) database management tools (i.e. MongoDB Compass) and Business Intelligence (BI) integration (i.e. MongoDB Connector for BI).

In terms of its design philosophy MongoDB with its Nexus Architecture strives to combine the crucial capabilities already present in most relational databases, namely: **an expressive query language**, **secondary indexes**, **strong consistency** and **enterprise management and integration**, with the innovations introduced by the NoSQL technologies: **flexible data model**, **scalability and performance** and **always-on global deployments** [97].

Concerning its data model MongoDB stores data as documents in a binary representation of JSON called BSON (Binary JSON). This representation is more efficient and presents lower parsing overhead when compared to JSON. BSON also allows for a richer support for data types such as dates and binary data. A document can be simply a flat table-like structure of key-value pairs or contain deeply nested arrays and sub-documents. No schema is imposed and there is no

need to declare the structure of documents to the system. The maximum BSON document size allowed is 16 Megabytes [94]. MongoDB also provides data governance support by providing document validation within the database and allowing the enforcement of check constraints on document structure, data types, data ranges and mandatory fields [98].

MongoDB provides its own Javascript-based query language supporting a rich range of query types and much like most relational databases includes the ability to perform joins, complex aggregations, text-based search and data transformations, amongst many others. MongoDB also supports graph traversals and MapReduce queries. A connector for Apache Spark is also available allowing for the use of ML to perform data analysis. Indexing is another important characteristic of relational databases taken into account. MongoDB includes support for many types of secondary indexes such as compound, unique, array, geospatial, sparse and text indexes. These can be declared on any field in the document, including fields within arrays.

MongoDB presents a plethora of features. Amongst its many capabilities MongoDB provides a multimodel architecture allowing multiple storage engines optimized for different workloads and operational requirements to be used within a single deployment, supports multiple sharding policies, can apply data compression in terms of physical storage and network transport, delivers fault tolerance and disaster recovery capabilities thru replica sets (i.e. multiple copies of data) and offers security and access control facilities such as authentication, authorization, encryption and auditing.

MongoDB is also very easy to integrate at the application level, providing native drivers for all popular programming languages and frameworks such as Java, JavaScript, .NET, Python, Perl and PHP as well as more than 30 community-developed drivers. This and more information can be found in [100,101].

2.5.3 In-Memory Databases

This subsection presents In-Memory Databases. The subsection starts by introducing this family of database related products, as well as some of the more relevant concepts associated with it. An example of such a database is also presented.

Disk IO (i.e. writing to and reading from a disk device) as always been regarded as an operation to be avoided as much as possible in the database systems world [93]. The reason for this is that disk IO has always been many orders of magnitude slower than CPU or memory access, being considered as the main bottleneck in database performance. Recently, the emergence of the Solid State Disk (SSD) technology as allowed for a huge leap in terms of disk IO performance.

SSD's however have some unique traits such as the write amplification effect (i.e. each write operation may cause multiple physical IO operations on the disk greatly impacting

performance) which prevent traditional relational databases to fully take advantage of this technology [93]. Many non-relational systems (e.g. Cassandra [99]) utilize different approaches such as log-structured storage engines, that are more suitable to the SSD technology performance characteristics [93].

The increasing capacity of servers in terms of available memory, presents yet another option, an in-memory solution: keep the whole database in memory and avoid IO altogether. This paradigm shift has led to the emergence of the so called in-memory databases [93].

Taking advantage of a large memory system, be it a single server or a cluster, however, requires a new architecture, aware to the fact that the whole database resides in memory, and that can provide the advantages of high speed access to data while also preventing data loss, in case of a power failure. In order to address these issues, the in-memory architecture presents two changes relatively to traditional database architecture [93]: a **cache-less architecture** and an **alternative persistence model**.

A *cache-less architecture* because in in-memory databases the data is already in memory and therefore does not need to be cached again in memory as it is the case in traditional disk-based databases in order to avoid disk IO. An *alternative persistence model* because the data disappears as soon as the system is powered off. Therefore some alternative mechanism must be in-place to preserve data.

To this end, several techniques are employed such as: replicating the data to other servers in the case of a cluster environment, writing complete images of the database to disk, also known as snapshots or checkpoints and writing transactions or single operations records into an append-only file (AOF) to disk, also called a transaction log or journal. In the last two cases, at system start up, the database can be restored or loaded back into memory using the files previously saved to disk. Redis [100] and VoltDB [101] are two examples of in-memory databases. Redis is presented next.

Redis

Redis (Remote Dictionary Server) is essentially an in-memory key-value store. In the key-value architecture, keys point to objects. In Redis these objects can be of any data type supported by Redis such as strings and various collections thereof (e.g. lists, sets, hashes). As is the case in the key-value family, Redis does not provide any kind of secondary indexing mechanism and only primary key lookups are supported. For data persistence, several options are provided [102]: **RDB persistence**, **AOF persistence**, a combination of **RDB and AOF** and **no persistence** at all.

RDB persistence creates database snapshots by generating an .rdb file on disk containing a copy of the whole database at a point in time. Snapshots can be created on demand or be

configured to occur on a scheduled basis or on given thresholds. *AOF persistence* keeps a journal of changes that can be used to 'roll forward' a database from a snapshot in case of failure or reconstruct the database on startup. Writes to the AOF file can be configured to occur for example after each write request to the server or at specified time intervals. *RDB and AOF* allows for both persistence models to be used in conjunction. In this case the AOF file will be used at server startup as it is guaranteed to be more up to date than the RDB snapshot.

Finally, in the case of *no persistence*, Redis will be used as a cache only, without any persistence of data. This means that all data will be lost on server shutdown. Both RDB as well as the AOF persistence models present advantages and disadvantages. RDB is more compact, maximizes performance and allows faster restarts. However RDB is not the right choice if data loss must be minimized. AOF on the other hand provides more guarantees of data persistence but can be slower than RDB.

Redis is not as sophisticated as other non-relational database systems such as MongoDB but still provides many useful built-in capabilities such as support for transactions, master-slave replication and several eviction policies (when used as a cache). Redis also provides high availability, that is, the ability to create Redis deployments capable of resisting to failures without human intervention (i.e, Redis Sentinel) and automatic partitioning by automatically sharding data across multiple nodes (i.e. Redis Cluster).

Integration with applications is also very easy, as Redis provides a very rich set of methods and operations. Redis can be accessed thru the command-line interface or via a wide range of client libraries, including support for Java, Python, Ruby, C, C++, Lua and many others. This and more information can be found in [100].

Summary

This chapter presented the state of the art regarding the scope of this work. Social Media was discussed in terms of its growth, its relevance as a topic of interest to several fields of research and also regarding the challenges that this type of data may present to such research. In terms of research topics, the field of event detection in Social Media data was addressed in more detail, as it is also the focus of this work. Machine Learning was also a topic of discussion due to its relevance in this field of research.

The related work was also presented, highlighting the main differences between these systems and the system proposed in this work. To better contextualize the remainder of the document, a review of the technologies used, as well as other relevant concepts related to the implementation of the system proposed, were also presented and discussed.

3 Conceptual Architecture

This chapter presents an overview of the system to be implemented in the scope of this dissertation. The chapter starts by presenting the definition of some concepts used throughout the remainder of this document in order to better define their meaning in the scope of this work. Next, a general overview of the system is discussed, concerning its main goals, the data to be used, the functional and non-functional requirements and also the main tasks identified according to those requirements. The architecture of the system is presented next, giving special emphasis to the presentation of the main blocks of the system and their responsibilities relatively to the main tasks identified previously. The discussion of the parameterization of the system concludes the chapter.

3.1 Definitions

This section presents several concepts used throughout this document. Some of these concepts were introduced in the proposal of Twevent, the system used as the base of the implementation. They are nevertheless presented here again for completeness. More details regarding those concepts can be found in the original paper in [10].

An **n-gram** is a general concept used to refer to a contiguous sequence of one or more words in a text. A more specific designation can also be used according to the size of the n-gram (the number of words that compose it), such as **unigram**, in the case of an n-gram composed of a single word (e.g. *car*), **bigram** in the case of two words (e.g. *my car*), **trigram** for the case of three words (e.g. *my new car*), etc.

A **segment** is similar to an n-gram in meaning except that when referring to a segment it is also implied that these n-grams could possibly refer to semantically meaningful units such as entities (e.g. the name of a person or a place) or collocations. *Cristiano Ronaldo* (a person) and *Figueira da Foz* (a place) are examples of segments. This concept was introduced in the base system.

A **tweet segment** is a concept introduced in the base system, to refer to a segment in a tweet message.

An **event segment** is a concept also introduced in the base system to refer to tweet segments considered to be possibly related to events. These consist of the top *K* ranked tweet segments obtained.

A **candidate event** is also a concept introduced in the base system to refer to a set of related event segments. For example the set of related event segments: *roller hockey*, *hockey*, *portugal*, *european champions*, *champions*, refers to a candidate event. This candidate event may ultimately be related to a real-world event as in this case (Portugal won the 52nd edition of the CERH European Roller Hockey Championship).

A **time window** denotes a period of time of fixed size (e.g. a day) used to group together tweets created in that time period. Events are identified inside these time windows. This concept was introduced in the base system.

A **sub-time window** is similar in meaning to a time window except that it denotes de fact that the time window is to be further sub-divided. As an example of this, if the size of a time window is set to be a day, and the size of the sub-time window is set to be 12 hours, then this time window can be sub-divided into 2 sub-time windows. Sub-time windows are used to capture the frequency patterns of event segments also used in the similarity calculations computed between these event segments. This concept was introduced in the base system and the usage just described is explained in more detail in Subsection 3.3.1 in the original paper.

Segmentation or **tweet segmentation** refers to the task of splitting the text of a tweet into non-overlapping segments (tweet segments). For example the tweet "*Cristiano Ronaldo scored a goal*" may for example be split into the following tweet segments: *Cristiano Ronaldo*, *scored*, *a* and *goal*.

User support of a tweet segment is a concept introduced in the base system, to refer to the number of unique users that created tweets containing that segment, within a given time window. The user support is used as a factor in the ranking scheme of the tweet segments. This ranking scheme is explained in more detail in Section 3.2 in the original paper.

A **Wikipedia anchor** is a link to some topic of interest, appearing in a Wikipedia article, which allows the user to visit another article specific to that topic.

A **newsworthy event** is a real-world event that would be considered of interest by news agencies or the Media. An example of this could be a football game or a political event.

The term **real event** is used to denote candidate events considered to be related to real-world newsworthy events. The candidate event previously referred to, namely the candidate event composed of the following event segments: *roller hockey*, *hockey*, *portugal*, *european champions*, *champions*, is an example of a real event.

The **probability of a segment** denotes the probability of finding that specific segment in a given corpus. In this case the corpus is composed by the tweets. This probability is used to compute the semantic meaningfulness of the segments.

The **Wikipedia anchor probability** of a segment denotes the probability that a segment occurs as an anchor in the Wikipedia articles where that segment also occurs. This concept was introduced in the base system. This probability is used to compute the newsworthiness of the segments.

The **segment frequency probability** denotes the probability of observing a given frequency (i.e. the number of tweets containing that segment) of that specific segment. This concept is introduced in the base system. This probability is used to compute the bursty probability of segments, also used as a factor in the ranking scheme of the tweet segments. This ranking scheme is explained in more detail in Section 3.2 in the original paper.

In the scope of this work the definitions of **event detection** and **event** used are the unifying definitions presented in Section 2.1.3. Specifically in this work, the set of real-world events to be detected or identified, should be those considered to be newsworthy. In terms of the information provided about these events, as alluded to in the definition of event detection, a textual representation of each of the events is provided by a set of representative and descriptive segments related to that event, in order to better describe and contextualize it. An action in this context is a post of new content (i.e. a new tweet).

3.2 Goal

The purpose of this work is to implement a segment-based event detection system to detect newsworthy events from tweets. These could be any real-world event of sufficient interest to the general public in order to be reported in the Media (e.g. newspapers, radio, television, online news, etc). Sport events (e.g. a football game), political events (e.g. elections) or musical events (e.g. summer concerts) are examples of potentially newsworthy events. Also no particular type of entity, such as celebrities or a specific place or location is targeted and all kinds of newsworthy events are considered as opposed to the systems proposed in [53, 55–58] and presented in the related work.

The detection of the events should be performed independently, considering separate time windows of fixed size. As an example, considering the size of these time windows to be a day, then the system would perform the detection of events for time window t_1 corresponding for example to the 1st of January of 2016 using only the tweets created during that time period and independently from the detection of events for time window t_2 corresponding for example to the 2nd of January of 2016. This also allows for further opportunities to parallelize and distribute the system as these

computations occur independently. This also means that the goal of this system is not to detect events in real time such as proposed in [24] in the related work.

Finally, each of the events identified should be represented by a set of its most representative segments, in order to provide as much information and context as possible about the event, so that a human operator can more easily interpret the results. In order to achieve this, these segments should therefore compose semantically meaningful units, of one or more words, as these are potentially more informative, when compared for example, with the use of just single words as proposed in [28].

3.3 Data

In terms of the source of data to be used, Twitter was chosen, for some of the reasons already mentioned in Section 2.1.2. Twitter was also chosen due to its common use as the data source in other similar systems proposed in the literature such as the ones presented in Section 2.4 when discussing the related work. Specifically for this work, the tweets should be written in the portuguese language as this language is also the focus of this work.

3.4 Requirements

The system to be implemented presents several requirements due to a few factors such as the challenges associated to the task to be performed (the detection of newsworthy events in this case), other challenges posed by the design choices and user usage of the chosen Social Media platform (Twitter in this case) and the nature of the data used. These requirements are discussed next.

3.4.1 Functional Requirements

In terms of its functional requirements the system should be able to fulfill the following requisites:

Requisite 1: The system should be able to detect only newsworthy events as much as possible.

This requirement stems from the task to be performed and as to do with some of the challenges associated with the chosen platform already presented in Section 2.1.4. The system should therefore be able to filter out from its results, events associated with non-newsworthy topics such as daily chatter and conversations and only present those more likely to be associated to real-world newsworthy events.

As an example of this, the event described by the following segments: *math*, *exams*, *flunk*, *grade*, *teacher* and *tomorrow*, possibly alluding to math exams taking place in some school the next day and doubtfully newsworthy, should not be considered. On the other hand, the event described by the following segments: *roller hockey*, *hockey*, *portugal*, *european champions*,

champions, alluding to the fact that the Portuguese national roller hockey team won the 52nd edition of the CERH European Roller Hockey Championship, clearly much more newsworthy, should be considered and presented in the final results.

Requisite 2: The detected events should be presented by descriptive segments associated to the events themselves, in order to give as much information and context as possible about them and also to facilitate their interpretation.

This requirement stems from the same reasons as before. As an example of this, let us consider the event presented earlier but now described by the following segments: *roller*, *portugal*, *european*, *hockey*, *champions*. This representation seems much less readable than the one previously presented. Also, abbreviations and disperse names should be avoided (e.g. *rui Patricio*, *penalty* as opposed to, for example *rui*, *penalty*, *patricio*). Ideally the human consumer of these results should not have to resort to additional information, such as for example the date reported on the event in order to infer its meaning.

Requisite 3: The requirements just presented, focus two aspects that should be taken into account regarding the qualities of the segments obtained by the system, namely: these segments should constitute semantically meaningful units as much as possible and also be potentially newsworthy.

In order to achieve this, these properties should each be quantified by a value that could then be used by an appropriate measure in order to rank the segments in this regard. These values in turn should be kept in a precomputed format allowing for their easy lookup. This is in order to avoid the cost of having to recompute these values every time a segment must be evaluated regarding these two aspects. One possibility to achieve this, is to use external services that provide one or more of these values already precomputed, in case such services exist. Another possibility may be to deploy an infrastructure in order to precompute and store these values in an appropriate format for later use.

Requisite 4: An appropriate infrastructure must also be in place, in order to allow for the pre-processing of the data source of the tweets as well as its storage in an appropriate format for later ease of access and retrieval. This infrastructure should also take into account the potentially high volume of data used.

3.4.2 Non-Functional Requirements

In terms of its non-functional requirements the system should be able to fulfill the following requisites:

Requisite 1: The system should be performant and scalable.

This requirement stems from the nature of the data used, a type of Big Data, whose inherit challenges were already discussed in Section 2.1.4. The system should therefore be as least costly as possible in terms of both the computational resources and the time required for it to perform its task. The system should also be scalable in order to account for the potentially high volume of data that must be processed. In this regard the design of the system should be flexible enough to allow for parallelization and possibly the use of distributed computing techniques.

Requisite 2: The system should be as accurate as possible in order to maximize the value of the data. The results obtained by the system, as measured by *precision* and *recall*, should be as close as possible to the ones obtained by the original implementation or similar ones, or ideally better.

3.5 Main tasks

From the requirements presented, the following main tasks were identified:

- the text of each tweet should be split into a set of non-overlapping segments, the so called tweet segments. These segments in turn should constitute semantically meaningful units as much as possible and should also be potentially newsworthy.
- the tweet segments obtained should then be filtered out in order to keep those more likely to be related to real-world newsworthy events, the so called event segments;
- the event segments obtained and related to the same event should be grouped together in the so called candidate events;
- the candidate events obtained should be filtered out in order to retain only those considered to be related to real-world newsworthy events, referred to as real events. These should be presented by their most representative event segments.
- an infrastructure to support the pre-processing and appropriate storage of the dataset in an appropriate format for later ease of access and retrieval should be setup and implemented;
- an infrastructure to support easy lookup of the required precomputed values should also be setup and implemented.

3.6 Architecture

This section presents the overall conceptual architecture of the system. First the system is presented as a whole and then its several different blocks are discussed in more detail. This detailed discussion aims to present the several components of these blocks and their respective responsibilities regarding the tasks identified.

In terms of its conceptual architecture, depicted in Figure 1, the system is composed of 3 main blocks: the **data source infrastructure**, the **precomputed values infrastructure** and the

event detection pipeline. Both of these infrastructures should be in-place before the system can be used. The event detection pipeline in turn represents the core functionality of the system and is composed of 4 main components: the **tweet segmentation** component, the **event segment detection** component, the **event segment clustering** component and the **event filtering** component. These components form the event detection pipeline used to identify the events in each time window independently. These blocks are discussed in more detail next.

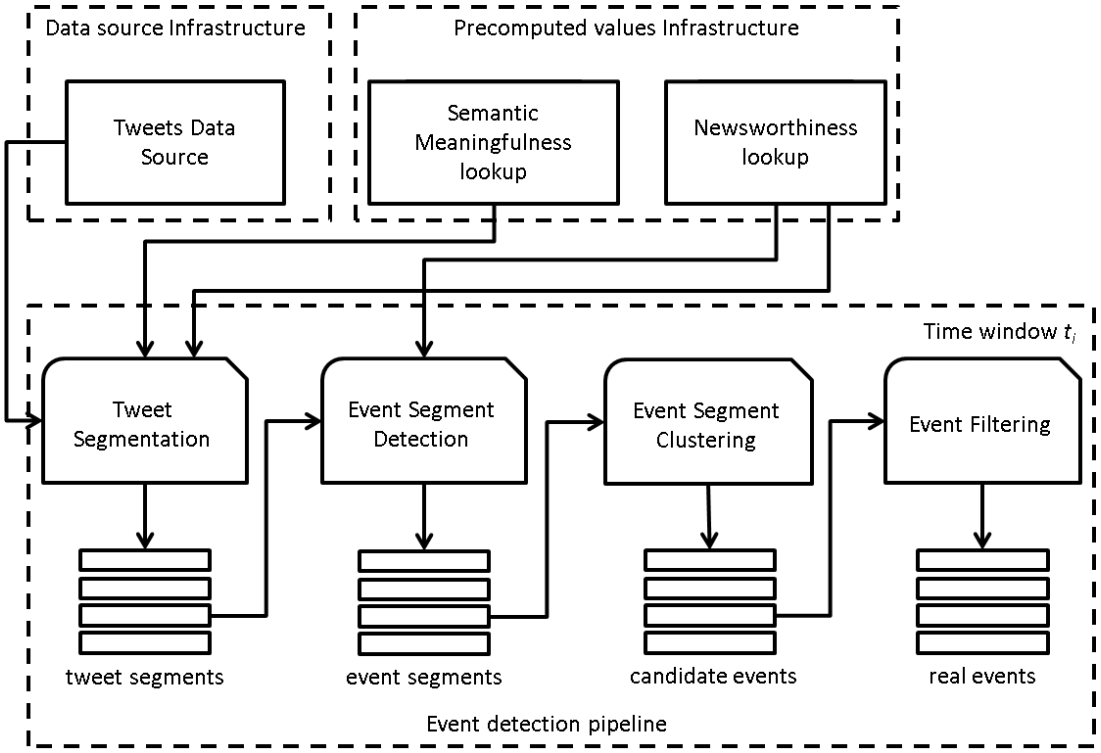

Figure 1: Conceptual architecture of the system.

3.6.1 Event Detection Pipeline

This subsection presents the event detection pipeline block, specifically in terms of its main components and the data workflow throughout this pipeline.

Given the main tasks just identified and to keep the design as flexible as possible, in order to allow for the use of parallelization techniques or even distributed computing, it was decided to separate and isolate these main tasks into different components, namely: the **tweet segmentation** component, the **event segment detection** component, the **event segment clustering** component and the **event filtering** component. These components should therefore form a pipeline as depicted in Figure 1, where data flows as it is processed along this pipeline and the output result of one component serves as the input of the next. These components are discussed next regarding their properties and responsibilities along this pipeline.

Tweet Segmentation Component

The goal of this component is to perform the first task identified. Its main responsibilities are therefore the following:

- **Input**: A set of tweets from a time window *t* obtained from the *Tweets Data Source* source. Lookup values for both the semantic meaningfulness and newsworthiness of the segments obtained from the *Semantic Meaningfulness lookup* and *Newsworthiness lookup* sources, see Figure 1.
- **Responsibility**: Find the most semantically meaningful and newsworthy segments in the text, using their lookup values. These values can be used as input to some appropriate measure in order to derive a unique value that quantifies these two properties.
- **Responsibility**: Use the aforementioned measure value computed for each segment and split the tweet into the combination of the segments that maximizes this overall measure;
- **Output**: A set of tweet segments.

As there are many possible combinations to split the tweet into segments, this segmentation task may then be regarded as an optimization problem, potentially very expensive to compute. The tweet '*Cristiano Ronaldo scored a hat trick*' for example may be split into the set of segments S_1 = {*Cristiano, Ronaldo, scored, a, hat, trick*} or the set of segments S_2 = {*Cristiano Ronaldo, scored, a, hat trick*}. This component should therefore implement a suitable algorithm in order to perform this task efficiently. Also, a suitable measure should be used in order to capture the concepts of semantic meaningfulness and newsworthiness.

Event Segment Detection Component

The goal of this component is to perform the second task identified. Its main responsibilities are therefore the following:

- **Input**: A set of tweet segments obtained from the *Tweet Segmentation* component. Lookup value for the newsworthiness of the segments obtained from the *Newsworthiness lookup* source, see Figure 1.
- **Responsibility**: Rank the tweet segments using an appropriate ranking scheme. This ranking scheme in turn should be supported by a measure of the potential association of the tweet segment with a real-world newsworthy event. The newsworthiness lookup value can be used in order to derive this measure.
- **Responsibility**: From this ranking, chose only the top *K*, as considering the full list of the tweet segments obtained would be unfeasible due to its potentially huge size in number;
- **Output**: A set of event segments.

This component should try to boost potentially more informative and newsworthy segments up the rank and keep the more common use and less informative ones as low as possible in that same rank. This is important due to the fact that only the top *K* of these segments are retained for further processing, therefore it is important to keep the most informative and newsworthy segments as high in the rank as possible in order to avoid them from being discarded.

Event Segment Clustering Component

The goal of this component is to perform the third task identified. Its main responsibilities are therefore the following:

- **Input**: A set of event segments obtained from the *Event Segment Detection* component, see Figure 1;
- **Responsibility**: Group the event segments that are related to the same event, using an appropriate measure of similarity;
- **Output**: A set of candidate events.

The algorithm used to compute these groupings should be deterministic, in order to always obtain the same results and should also be as efficient as possible due to the large number of event segments that must be pairwise compared to compute their similarity. Ideally this algorithm should be non-iterative to yield the best performance. An appropriate measure of similarity should also be derived so that unrelated and therefore different events can be differentiated and captured by different candidate events. Similar events (e.g. two football matches) occurring at the same time should also be differentiated and captured in separate candidate events as much as possible. This issue should also be taken into account when deriving similarity.

Event Filtering Component

The goal of this component is to perform the fourth and last task identified. Its main responsibilities are therefore the following:

- **Input**: A set of candidate events obtained from the *Event Segment Clustering* component, see Figure 1;
- **Responsibility**: From the list of candidate events obtained, retain only those considered to be related to real-world newsworthy events and discard all the others;
- **Responsibility**: Present each of the retained events by its textual representation. This textual representation should be composed by the most representative and informative segments related to the event in order to allow for its easy interpretation by a human operator.
- **Output**: A set of candidate events related to real-world newsworthy events, or real events.

As the number of candidate events obtained may potentially be large, with most of these not being related to real-world newsworthy events, this component should be as accurate as possible in terms of both *precision* and *recall* when performing the filtering step. This filtering step can be performed by using some kind of threshold value, obtained empirically or by some other means.

Another option is to try to capture the distinctive features that potentially explain the reason why a candidate event is related or not to a real event and use these distinctive features to perform the filtering step. Machine Learning techniques for example, can be used to achieve this. The textual representation of the events obtained should also be as descriptive as possible. The segments should therefore be presented ordered by some measure able to capture their informative potential relatively to the event.

3.6.2 Data Source Infrastructure

The data source infrastructure should allow for the proper pre-processing of the dataset (i.e. the tweets) used to perform event detection. This infrastructure should also allow these pre-processed data to be stored in an appropriate format for later ease of access and retrieval. This is denoted as the *Tweets Data Source* source in Figure 1. A database store can be used for this purpose.

The database used should be adequate to the format of the data and also be prepared to work with a potentially high volume of data. The database should also allow indexing for better querying performance and implement useful operations for the pre-processing of the data, such as aggregation, averaging and counting. This infrastructure should be in-place before the system can be successfully run.

3.6.3 Precomputed Values Infrastructure

The precomputed values infrastructure is composed of 2 components, namely the sources of the precomputed values mentioned earlier in Section 3.4.1, concerning the semantic meaningfulness (*Semantic Meaningfulness lookup*) and newsworthiness (*Newsworthiness lookup*) of the segments respectively, see Figure 1. This infrastructure should therefore allow for the computation and storage of these values so that they can be easily looked up later on.

Again a database store may be used for this purpose. The only storage needs required are for the segment and the respective measure value for its semantic meaningfulness and newsworthiness. This may also be accomplished by using external services if possible. In case these values cannot be computed for all the possible segments, an appropriate strategy should be devised in order to deal with unknown segments. These values could even and ultimately be

learned as the system performs new detections (this option was not explored in this work). This infrastructure should also be in-place before the system can be successfully run.

3.7 Parameterization

In terms of the parameterization of the system, two parameters were identified so far, namely the size to use for time window *t*, determining the set of tweets to be used during the segmentation phase performed by the Tweet Segmentation component and the top *K* tweet segments to retain as event segments, obtained as the output of the Event Segmentation Detection component. Depending on the implementation choices of the components identified, this list may grow accordingly. The full list of parameters is therefore presented in the next chapter during the discussion of the implementation of the system.

Summary

This chapter presented the conceptual architecture of the system. The requirements were identified and from these, the main tasks were derived. Two of these requirements have to do with the need to obtain the best segments, in terms of both their potential semantic meaningfulness and newsworthiness. In order to fulfill these requirements, three main blocks were identified, namely the Data source Infrastructure, the Precomputed values Infrastructure and the Event detection pipeline.

The several components composing each of these blocks were then discussed in more detail, regarding their goals and main responsibilities, concerning the main tasks outlined. In this regard, the four components composing the Event detection pipeline, namely, the Tweet Segmentation component, the Event Segment Detection component, the Event Segment Clustering component and the Event Filtering component, were subject to a thorough analysis. The parameterization of the system concluded the discussion.

4 Proof of Concept

This chapter discusses the implementation details of the proposed system, based on the conceptual architecture outlined in the previous chapter. The chapter starts by presenting some generalities about the implementation of the system, such as the language used, the reference systems considered and the choice of the dataset. The architecture of the system is then presented in order to map the conceptual architecture to the several blocks implemented. The implementation details of the main components of the system, composing the Event detection pipeline, are then presented. This presentation also aims to map the requirements and main responsibilities identified for each of these components, to the implementation choices considered.

The infrastructure implemented in order to manage the several precomputed values required by the system, some of which were identified in Section 3.4.1, is presented next. The parameterization of the system is discussed at the end of the chapter. Regarding the infrastructure implemented to support the pre-processing and storage of the dataset, this topic is discussed in the next chapter in full detail.

4.1 General Overview

This section presents some implementation generalities, such as the language used to implement the system, the system used as the base of the implementation and the data source chosen.

4.1.1 Implementation Language

The language used to implement the system was Python 3.5, mainly due to its rich ecosystem, particularly in terms of packages available for data science and also due to its easy integration with many third-party products such as for example most of the existing databases. Python was introduced in Section 2.5.1.

4.1.2 Reference Systems

As previously discussed in Section 1.2, the system implemented uses as the base of its implementation a similar system proposed in [10]. The reasons for the choice of this specific system were also discussed in the aforementioned section. One of those reasons was the fact that the system is well defined and formalized by its authors. For this reason, some of the implementation details are therefore left out and not presented in this document. These details can be found in the original paper. Another system was used as reference, namely the system proposed in [11]. Regarding this system however, only the features used to train the SVM model were used as reference. Section 4.3.4 discusses this topic in more detail.

4.1.3 Data

As already mentioned Twitter was used as the data source of the system and the reasons for this choice were also presented in Section 2.1.2. More concretely the dataset used consists of a set of tweets collected as part of the TVPulse [53] project, a project developed at Instituto de Telecomunicações de Aveiro to detect TV Highlights using Social Media data. The rationale behind the choice of this particular dataset, as well as its analysis, are the main topics of discussion of chapter 5.

4.2 Proof of Concept

In terms of the implementation of the blocks identified in Section 3.6, the following decisions were made:

- in terms of the Precomputed values Infrastructure block, the Semantic Meaningfulness lookup source is provided by the Segment probability source and the Newsworthiness lookup source is provided by the Wikipedia anchor probability source. The motivation behind the need for these lookup sources was discussed during the presentation of the conceptual architecture. Due to specific aspects concerning the implementation of the base system, discussed in the original paper in Section 3.2, a third precomputed value, namely the Segment frequency probability, was added, see Figure 2. All of these values are stored in a Redis database instance. The process used to compute these values as well as the rationale behind the choice of Redis as their storage backend, are presented in more detail in Section 4.4.
- concerning the Data source Infrastructure block, MongoDB was chosen as the storage backend for the tweets used by the system as its data source, see Figure 2. This topic is discussed in more detail in chapter 5, along with the preparation, analysis, cleanup and storage of these data.

- regarding the Event detection pipeline block, its main components remain unchanged as well as the workflow of the data along this pipeline as already mentioned in Subsection 3.6.1. This is depicted in Figure 2. The internal processes of each of these components are also detailed, namely: the **segmentation algorithm** in the Tweet Segmentation component, responsible for segmenting the tweets into their respective tweet segments; the **weighting scheme** in the Event Segment Detection component, responsible for ranking the tweet events in order to retain only the top *K* as event segments; the **Jarvis-Patrick** clustering algorithm in the Event Segment Clustering component, responsible for grouping together the related event segments into candidate events and the **SVM model** in the *Event Filtering* component, responsible for filtering the candidate events to retain only those related to real-world newsworthy events or real events.

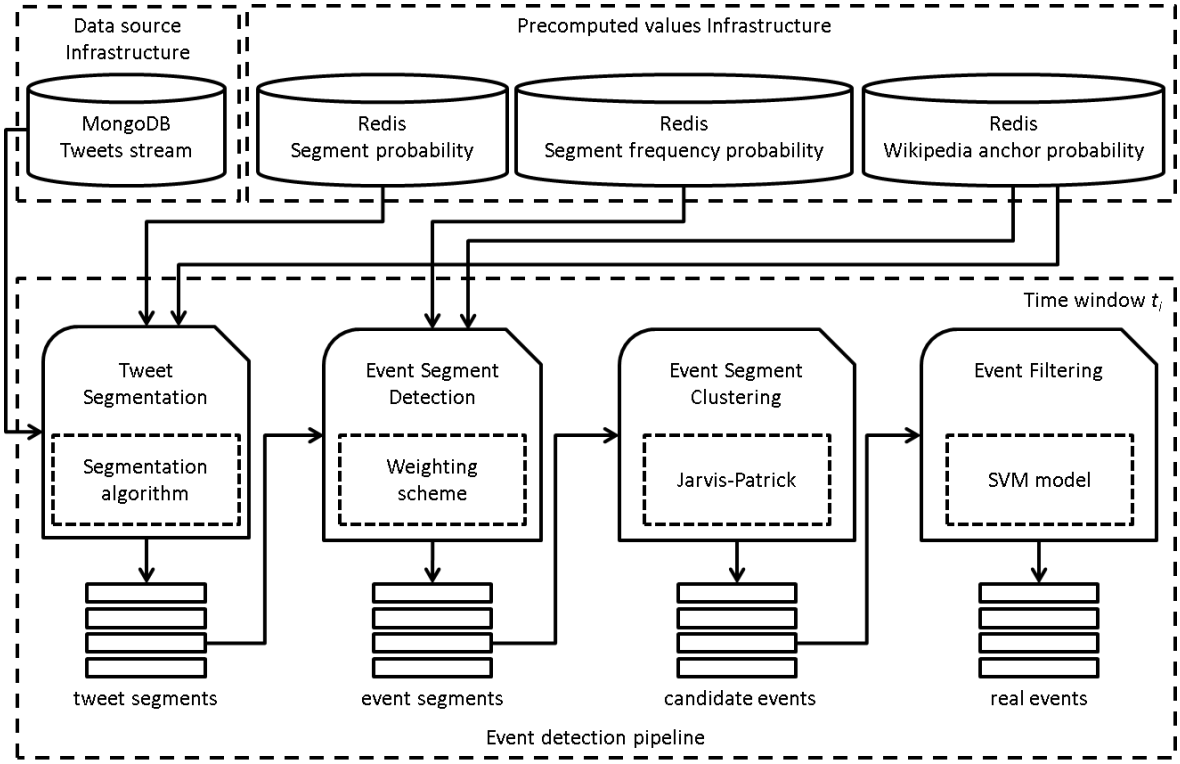

Figure 2: Implemented architecture.

4.3 Event Detection Pipeline

This section discusses the implementation of the Event detection pipeline, specifically the implementation of its four main components. The main reasons behind the implementation choices made, are also discussed as appropriate and mapped to the requirements and responsibilities previously identified during the presentation of the conceptual architecture of the system in Chapter 3.

4.3.1 Tweet Segmentation Component

As already mentioned, the goal of this component is to partition a tweet into a set of non-overlapping and consecutive segments (i.e. a single word or a sequence of words), the so called tweet segments. The responsibilities and main goals of this component were already discussed in Subsection 3.6.1.

In order to achieve this in an efficient way, the authors of Twevent define this segmentation task as an optimization problem that aims to maximize the so called stickiness or cohesion of segments (i.e. find the best partition boundaries without breaking the segment as a unit) and then propose dynamic programming to solve it. While being just a guideline, as there are certainly other methods to solve such a problem, this was also the strategy followed in the implementation of this component.

It should be noted that for the purposes of this implementation only n-grams up to order 3 that is $n = \{1, 2, 3\}$ (i.e. unigrams, bigrams and trigrams) where considered as possible segment candidates. This represents a trade-off. On one hand not considering n-grams of higher orders (i.e. $n > 3$) can potentially exclude informative and descriptive segments for the purposes of event detection, as these are not considered in the analysis. On the other hand given the informal nature of Twitter (e.g. the common use of informal abbreviations) as analyzed in Section 5.2, it is not expectable to see such long sequences being used very often.

Another aspect to be taken into account has to do with the memory and storage constraints of the virtual machine where the system is to be deployed. Considering all possible n-gram orders would incur a cost too high in terms of both memory and storage space. Given these aspects it was then decided to consider only n-grams up to order 3.

In terms of the segmentation task itself, the segmentation algorithm implemented followed a dynamic programming approach as already stated. This was achieved by first considering the n-grams as a set of nodes composing a Directed Acyclic Graph (DAG). An example of this is depicted in Figure 3, where the tweet "*my car is fast*", is decomposed into all of its possible n-grams.

Each of these n-grams in turn is considered as a node and these nodes are linked together by directed edges according to the position in which they occur in the text. As there are no cycles in this directed graph and the order in which the nodes are linked together follows from the position in which the corresponding n-grams occur in the text, it is then possible to linearize this graph as shown in Figure 4, depicting again the same tweet. Under these circumstances, that is, by stating the problem in terms of a DAG, it is then possible to solve the segmentation problem using dynamic programming.

More formally, given a DAG $G = (V, E)$ where V denotes the set of vertices or nodes of the DAG and E the set of its edges, two more special nodes named *start* and *end* are defined and linked accordingly to the other nodes. The cost $Cost(e_i)$ of each directed edge $e_i \in E$ linking vertices u and v (i.e. $(u,v) \in V$) is then computed using Equation 9 (a modified version of Equation 3 in the original paper), where the segment s considered and used for this computation, is the one corresponding to the end node of the edge (v in this case).

Under these conditions the optimum segmentation can be solved as the maximum cost path search between the node *start* and the node *end*. In the case of the special nodes *start* and *end*, the cost of their incoming edges *Cost*(*end*) and *Cost*(*start*) is set to zero. Listing 1 presents the pseudocode of the iterative version used to find this maximum cost path, where *Cost(u)* denotes the total cost to reach node u and *Cost(v)* denotes the cost of edge e_i linking u to v and computed as described above. The Python code snippet of the implementation of this pseudocode is presented in Appendix A, Listing 15.

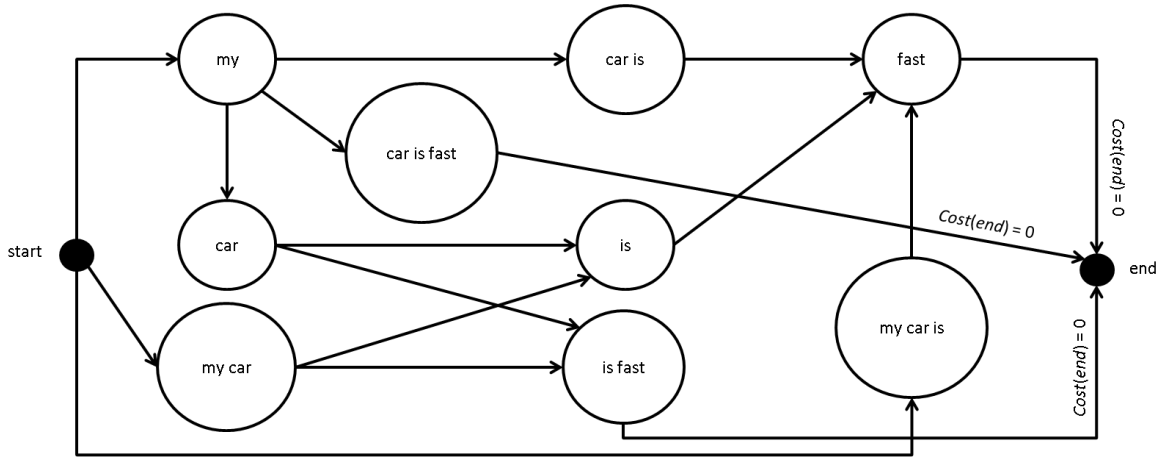

Figure 3: Representation of the text of a tweet (i.e. 'my car is fast') as a DAG of n-grams.

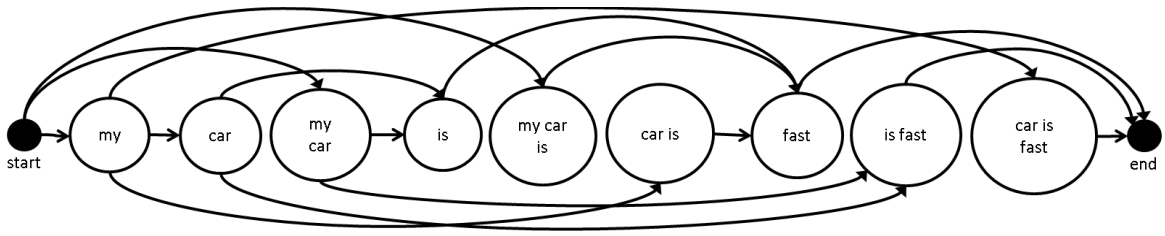

Figure 4: Linearization (topological ordering) of the DAG.

$$C(s) = L(s) * e^{Q(s)} * SCP(s) \quad (9)$$

$Cost(start) = 0$
for $v \in V \setminus \{start\}$ *in linearized order*
  $Cost(v) = max_{(u,v)\in E}\, Cost(u) + Cost(v)$

Listing 1: Pseudocode to find the maximum cost path using an iterative version.

It should be noted that the modification mentioned in Equation 9 is due to the fact that the method of computation used to derive the prior probability for segments as denoted by *P(.)* in Equation 8 (*Pr(.)* in Equation 2 of the original paper) was not the same used originally (i.e. the Microsoft Web N-Gram service). Therefore the *log* and the subsequent *sigmoid* were removed from the original formula.

The details concerning the setup of the infrastructure necessary to provide this functionality (i.e. derive the probability for segments) as well as the process used to compute the values for *P(.)*, are presented in Section 4.4.2. The formula for *SCP(s)* in Equation 9 (Equation 2 of the original paper) is therefore computed as depicted in Equation 8 for *n*-grams with $n > 1$ and as $SCP(s) = \mathrm{P}(s)$ when $n = 1$. The full details regarding Equation 9 can be found in Appendix B or alternatively in Section 3.1 of the original paper.

Prior to this segmentation step, the text of the tweets is first normalized in order to remove user mentions, links, hashtags and emoji. The text is also stripped of any accentuation and lowercased. Stripping words of their accentuation is a trade-off with possible implications in the results and is further discussed in Section 5.2.3. It should be noted that stopwords are not removed at this stage. This is done in order to avoid obtaining false n-grams such as for example in the case of the text "*Mary and Joan* went to Paris", where removing the stopword *and* would result in the false n-gram *Mary Joan* being computed. Figure 5 depicts this normalization pipeline.

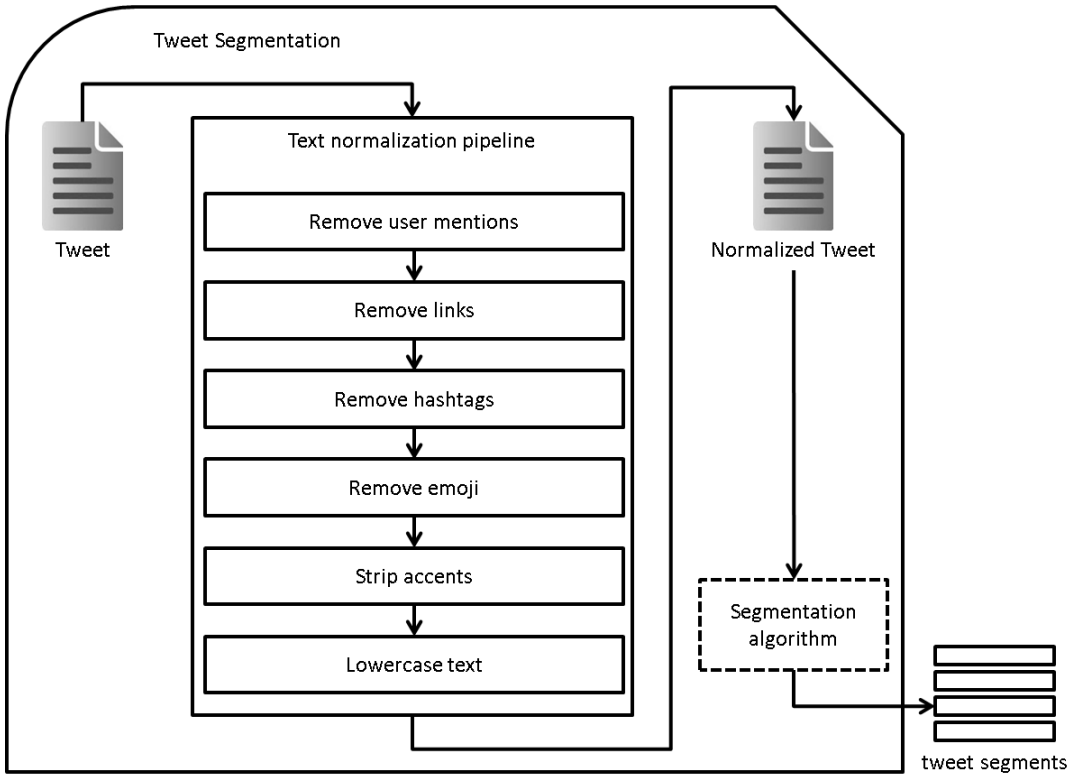

Figure 5: Tweet text normalization pipeline.

This was also done to preserve the names of locations for example. One such case would be the text "*I live at Figueira da Foz*" where removing the stopword *da* would result in the name of the location, *Figueira da Foz* in this case, to be changed to *Figueira Foz*, which is much less readable and not the proper name of the location.

Regarding the requirements stated during the conceptual discussion of the system, the elements of Equation 10 map as follows: the $e^{Q(s)}$ element of the equation is intended to use Wikipedia, more specifically the Wikipedia anchor probability of the segments, in order to boost those occurring more often as anchors in Wikipedia articles, as these are considered to be potentially more newsworthy. The $SCP(s)$ element is used to try to find semantically meaningful segments and is computed as already discussed.

In this regard it should be mentioned that several measures exist in order to try to find these semantic units, such as collocations as already presented in Section 2.3.3. These measures however present advantages and also disadvantages that make them more suitable or not, depending on the intended usage scenario. This also makes it harder to choose one in favor of the remaining. SCP was chosen as it seemed to be more general purpose (i.e. not just focused in finding collocations) and obtained good results in the tests conducted in [52] in order to assess its performance.

4.3.2 Event Segment Detection Component

Regarding the implementation of this component, this work proposes a change in the calculation of the weighting scheme used to rank the segments. This change is discussed next. As a quick remainder it should be noted that the goal of this component is to rank segments according to a weight scheme. This rank is then leveraged in order to select only the top K, also called event segments, for further processing. The responsibilities and main goals of this component were already discussed in Subsection 3.6.1.

In order to achieve this, a weight w_b is computed for each of the segments according to Equation 10 (Equation 8 in the original paper). In this equation $P_b(s,t)$ denotes the bursty probability of segment s in time window t and the $u_{s,t}$ term denotes the user support (i.e. the number of users that created tweets containing that segment) of that same segment in the same time window. The full details of this equation and its meaning can be found in Appendix C or alternatively in Section 3.2 of the original paper.

This work proposes the use of the Wikipedia anchor probability of the segment, denoted as *Q(s)*, as an additional factor in this computation. This change was also introduced in order to address the goals proposed for this component as stated in Subsection 3.6.1.

$$w_b(s,t) = P_b(s,t) * \log(u_{s,t}) \quad (10)$$

The rationale for this change stems from the fact that commonly used words are in general boosted by their usually greater user support when compared to other potentially more informative words, not as commonly used. As an example, Table 1 presents the list of the top 10 ranked segments (along with the counts for their user support) computed for three randomly chosen days, namely the 11th, 14th and the 24th of June of 2015. This ranking was computed using the formula originally proposed to compute the weight of the segments and presented in Equation 10 (Equation 8 in the original paper), where only the bursty probability of the segment and its user support are taken into account.

As it can be seen, the ranking positions of the segments seem to follow the same pattern for the three depicted days, that is, the segments with a greater user support are ranked higher. The only noticeable exception to this pattern occurs on the 14th and is highlighted in bold. Furthermore none of the segments listed is of particular interest in terms of the information it can potentially convey to the event detection process.

Swear words were elided from the listing and are denoted with the * symbol instead. The segments are listed top down according to their position in the rank (i.e. the first element in the list is ranked 1, the second is ranked 2, and so on). The count for the user support of each segment is presented in the column to the right of the respective segment (e.g. the user support for segment *amanha* for the 14th is 1,750).

Table 1: List of the top 10 ranked segments computed for three random days.

2015-06-14		2015-06-24		2015-06-11	
amanha	1,750	es	1,614	amanha	1,922
ver	1,714	sei	1,402	ter	1,678
vai	1,420	sempre	1,388	sei	1,488
dormir	1,328	bue	1,228	tudo	1,412
*	1,186	melhor	1,228	nada	1,362
assim	1,108	mim	1,152	mim	1,170
tempo	1,055	acho	1,118	*	1,100
exame	**994**	ti	1,102	assim	1,013
fds	**979**	aqui	1,042	porque	1,008
ta	**1,082**	nunca	909	escola	973

Intuitively it is in fact expectable to find that more commonly used words have a much greater user support than less common words, simply because they are used on a daily basis, and while taking the *log* may somewhat attenuate this effect, it should be noted that such common words can easily reach a user support in the order of the thousands, as opposed to less commonly

used words, where such a user support would only be expectable in specific situations, such as worldwide big events for example.

The result of this is that the second factor (i.e. user support) will dominate the weight result most of the time and thus boost the more common words (and also usually less informative due to their general purpose and common use) up in the rank. Considering that from these only the top K are retained for further processing, this can potentially mean the exclusion of many informative words from further analysis. In terms of tweet analysis this can become even more problematic, as much of the topics discussed are about personal and trivial matters, which in turn means that most common words used daily will be heavily used.

One possible solution to attenuate the issue just discussed would be to introduce a penalty scheme applicable according to the commonness of a segment. In such a scheme more commonly used words would be more penalized and therefore pushed down the rank. Another possibility and the one chosen in this case, is to leverage Wikipedia again in a similar way to what was done previously in the tweet segmentation phase.

More specifically, segments are boosted according to their Wikipedia anchor probability, denoted as *Q(s)* (i.e. the probability that a segment occurs as an anchor in the Wikipedia articles where it also occurs). This means that segments appearing more often as anchors (i.e. links to other articles) in Wikipedia and therefore also more likely to be informative in terms of event detection, will potentially be boosted up in the rank. More common use words on the other hand, are expected to drop down in the rank, as these words are not expected to be often used as anchors in Wikipedia articles and will therefore not benefit from this boosting factor.

The revised formula used to compute the weight of the segments is depicted in Equation 11. It should be noted that for segments where *Q(s) = 0* (i.e. segments that never appeared as anchors in Wikipedia) there is no change in the value obtained as compared to the former formula, as $e^0 = 1$.

$$w_b(s,t) = P_b(s,t) * \log(u_{s,t}) * e^{Q(s)} \quad (11)$$

Table 2 presents the list of the top 10 ranked segments (along with the counts for their user support) computed for the same three days as before, using the new proposed weight scheme. From the analysis of this listing, two facts stand out: 1) the top ranking no longer seems to follow the pattern mentioned before (i.e. segments being ranked mostly according to their user support) and 2) the occurrence of more informative segments in some cases, such as *neymar*, *brasil*, *david luiz* and *rui vitoria* for example.

Table 2: List of the top 10 bursty segments computed for the same three random days using the proposed weighting scheme.

2015-06-14		2015-06-24		2015-06-11	
neymar	247	ganda	310	raio	42
brasil	305	sdds	191	twitter	286
portugal	516	ask.fm	17	mase	117
david luiz	39	cristiano araujo	39	christopher lee	39
peru	76	es	1,614	matematica	168
mase	128	bue	1,228	rui vitoria	56
colombia	59	sei	1,402	portugues	223
meo arena	39	sempre	1,388	autocarro	82
portugues	230	bora	184	amanha	1,922
amanha	1,750	melhor	1,228	ter	1,678

4.3.3 Event Segment Clustering Component

The goal of this component as outlined in Subsection 3.6.1, is to cluster related event segments into candidate events. To compute these candidate events, that is, to cluster the event segments, a variant of the Jarvis-Patrick algorithm, taking only its *k* parameter into account (i.e. the number of nearest neighbors to examine for each point), was implemented, similarly to what is proposed in the original paper.

The reason to use this clustering algorithm, has to do with the fact that it is a non-iterative algorithm and therefore more efficient, as the clusters can be computed in a single pass, as opposed to other clustering algorithms such as K-Means which is iterative. This algorithm is also deterministic and therefore is a suitable choice concerning the requirements discussed in Subsection 3.6.1. Concerning the remainder of the considerations also made in this Subsection, namely regarding the fact that unrelated events as well as similar events occurring at the same time should be detected as separate events as much as possible, the original formulation (Equation 9 and 10 in the original paper) already addresses those issues.

The pseudocode of the implementation of this Jarvis-Patrick variant is presented in Listing 2. Segments are represented by their indices in the pairwise similarity matrix denoted by *pairwiseDMatrix*. These indices are then sorted in ascending order according to their similarity scores, using the *argsort* function and the last *k* of these indices are retained for each of the elements (i.e. the k-nearest neighbors *kn* of each segment). The *argsort* operation followed by the retention of the last *k* indices (higher similarity scores) is depicted in Figure 6 ($k = 2$ is used as an example only).

Each segment (e.g. column) is then tested sequentially in order to verify if it also occurs as a k-nearest neighbor of its k-nearest neighbors. If this is the case, then both segments are clustered together (*idx* and *idj* are the indices of the segments). Due to this sequential testing a small optimization can be introduced in order to avoid retesting indices (i.e. those smaller than the one being currently tested (the *continue* statement)). *nEls* denotes the number of columns in the pairwise matrix (i.e. the number of segments). The Python code snippet of the implementation of this pseudocode is presented in Appendix A, Listing 16.

```
Kn = argsort(pairwiseDMatrix)
Kn = choose only last k indices of Kn
for idx in range(nEls - 1)
   for idj in kn[idx]
      if idj < idx
         continue
      if idx in kn[idj]
         cluster idx and idj together
```

Listing 2: Pseudocode to implement the Jarvis-Patrick variant.

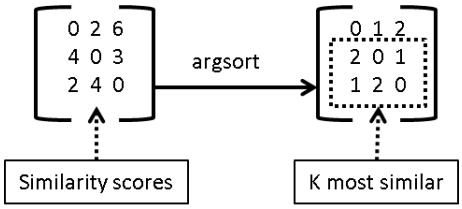

Figure 6: Argsort operation.

Finally it should be noted that in order to compute the TF-IDF representation of the pseudo-documents and the cosine similarity measure, both of which are used in the formula to compute the similarity between the pairs of segments (Equation 9 in the original paper), the facilities provided by the Scikit-Learn package were leveraged, specifically and respectively the TfidfVectorizer and the cosine_similarity classes. The full details concerning this similarity function can be found in Appendix D or alternatively in Section 3.3 of the original paper.

Regarding the TF-IDF calculation, the TfidfVectorizer class was first fitted with the dataset data in order to learn the vocabulary and then pickled (i.e. persisted in a binary format in a file) to be used for the TF-IDF calculations later on. Section 3.3 in the original paper details the similarity formulation used as well as its rationale.

4.3.4 Event Filtering Component

This subsection presents the implementation of the event filtering component. The subsection starts by outlining the rationale behind the choices made concerning this implementation, namely the choice of an SVM model in order to perform the filtering step. Next the process used to choose the

features as well as the tuning, training and testing phases of the SVM model are presented. The method used to manually label the data used for these purposes is however not addressed and is presented during the discussion of the tests performed to evaluate the performance of the system.

As already outlined during the discussion of the conceptual architecture of the system, the goal of this component is to perform the final filtering step in order to filter the candidate events obtained in the previous step and from these retain only those related to real-world newsworthy events, also referred to as real events. In the original proposal this step is performed via the use of a user defined threshold τ.

The reason to choose a trained model to perform this filtering step, as opposed to using a threshold, as proposed in the original system, as to do with two main aspects. First, in the experiments conducted by the authors of the original paper, it was found that the value of the parameter τ (i.e. the parameter used as the threshold value in the filtering step) affected the accuracy of the system more significantly than the other parameters. This means it had to be adjusted in order to find its most suitable value.

This task is however not trivial and usually implies some kind of deliberate trade-off, which in turn means that the value chosen is often not the optimal one. Performing such informed decisions also implies a deeper knowledge of the data. Secondly, using a threshold, as opposed to a pre-trained model, adds to the burden of the parameterization of the system, as a greater number of parameters must be setup upfront.

To help mitigate some of these issues a pre-trained classification model can be used as proposed in [11]. To do this, first a good set of features, able to capture the most distinctive characteristics of the computed candidate events, in terms of their structure as well as the features of its segments, must be chosen. These features can then be used to train the model. If the model obtained after training possesses a suitable generalization power (i.e. is able to correctly classify yet unseen data), it can then be used to perform the filtering step.

The use of a model does not come without its own issues, however. Specifically and as an example, in order to train the model, first a suitable training dataset must be obtained. As there is no ground truth that can be used in this case (i.e. a set of pre-labeled instances), this training data must be labeled. This in turn implies the labor intensive and time consuming task of manually inspecting and properly labeling possibly thousands of samples (i.e. candidate events). This can also be challenging as the quality of the labeling can profoundly affect the performance of the classifier.

Concerning the choice of SVM as the model to be used, the main factors taken into consideration were the following:

- the expected imbalanced nature of the training dataset. To understand this point let us define two classes, respectively: class *T* to denote a candidate event as being related to a real-world newsworthy event and class *F* to denote otherwise. The filtering step can then be seen as a binary classification problem, were candidate events are classified as belonging to either class *T* or *F* accordingly. However, it is expected that most of the candidate events are not related to real-world newsworthy events, which means that the training dataset will be imbalanced, that is, the majority of the samples (i.e. the candidate events) will belong to class F. This is problematic as classifiers generally perform poorly when trained with imbalanced datasets [103]. While SVM is not immune to this problem, some of its characteristics, such as the fact that it only takes into account the instances near the boundary (i.e. the support vectors) to build its model, therefore not taking into account instances far away from this boundary (i.e. the instances labeled with class *F* in this case) irrespective of their number, can potentially make it a bit more resilient to this issue [103].
- the fact that different kernels can be used with SVM meaning that even if for example the data is not linearly separable in the base feature space, SVM can still perform well via the use of an appropriate kernel.

All the details associated with the setup of this model, namely the selection of the features to be used and the tuning, training and testing phases are presented next.

Feature Selection

In terms of the features used to train the model, these consist of a subset of the statistical and social features proposed in [11]. In this regard it should be noted that the textual features proposed were not used as they involved using information embedded in the hashtags content and this work does not intend to use specific information, such as the one derived from hashtags or user mentions in order to perform the event detection. These features are formally defined and described in Table 3.

In order to choose the most discriminative features amongst these, a RandomForestClassifier (implemented by Scikit-Learn) was used with 80% of the training data to assess their respective importance. As depicted in Figure 7, the three features considered more relevant were respectively the *wiki*, *sim* and *tag* features, with the *rt* feature being considered by far the least relevant of all.

Given these results it was deliberated whether the *rt* feature should be used or not. Removing this feature might on one hand have as a consequence the loss of information. On the other hand, maintaining it may introduce unwanted noise. In the end it was decided to keep all features, in order to preserve as much information as possible. The Python code snippet used to find the importance of these features is presented in Appendix A, Listing 17.

Table 3: List of candidate features used to train the SVM model.

Feature	Description	Formula
seg	Ratio concerning the number of segments of a cluster or candidate event *e*. *Gset(t)* denotes the set of candidate events *e* computed in time window *t*, S_e denotes the set of segments of *e* and $\|S_e\|$ the number of these segments.	$\frac{\|S_e\|}{\max_{e' \in Gset(t)} (S_{e'})}$
edge	Ratio concerning the number of edges of a cluster or candidate event *e*. E_e denotes the set of edges of *e* and $\|E_e\|$ the number of these edges.	$\frac{\|E_e\|}{\max_{e' \in Gset(t)} (E_{e'})}$
wiki	Average newsworthiness of *e*.	$\frac{\sum_{s=1}^{\|S_e\|} Q(s)}{\|S_e\|}$
sim	Average similarity of the edges of *e*. *g* denotes an edge and $sim_t(s_a, s_b)$ is computed as shown in Equation 9 in [10], where s_a and s_b denote the two segments linked by edge *g*.	$\frac{\sum_{g=1}^{\|E_e\|} sim_t(s_a, s_b)}{\|E_e\|}$
df	Percentage of tweets related to *e* (i.e. tweets containing at least one segment of S_e and denoted as *T(e)*) relative to all tweets created in time window *t*.	$\frac{\|T(e)\|}{N_t}$
udf	Percentage of users related to *e* (i.e. users that posted tweets containing at least one segment of S_e and denoted as *U(e)*) relative to the number of users U_t who posted tweets in time window *t*.	$\frac{\|U(e)\|}{U_t}$
rt	Percentage of tweets that are retweets in *T(e)*. $T(rt)$ denotes the subset of tweets from *T(e)* that are retweets.	$\frac{\|T(rt)\|}{\|T(e)\|}$
men	Percentage of tweets with user mentions in *T(e)*. $T(men)$ denotes the subset of tweets from *T(e)* that contain user mentions.	$\frac{\|T(men\|}{\|T(e)\|}$
rep	Percentage of tweets that are replies in *T(e)*. $T(rep)$ denotes the subset of tweets from *T(e)* that are replies.	$\frac{\|T(rep)\|}{\|T(e)\|}$
url	Percentage of tweets containing url links in *T(e)*. $T(url)$ denotes the subset of tweets from *T(e)* that contain url links.	$\frac{\|T(url)\|}{\|T(e)\|}$
tag	Percentage of tweets containing hashtags in *T(e)*. $T(tag)$ denotes the subset of tweets from *T(e)* that contain hash tags.	$\frac{\|T(tag)\|}{\|T(e)\|}$

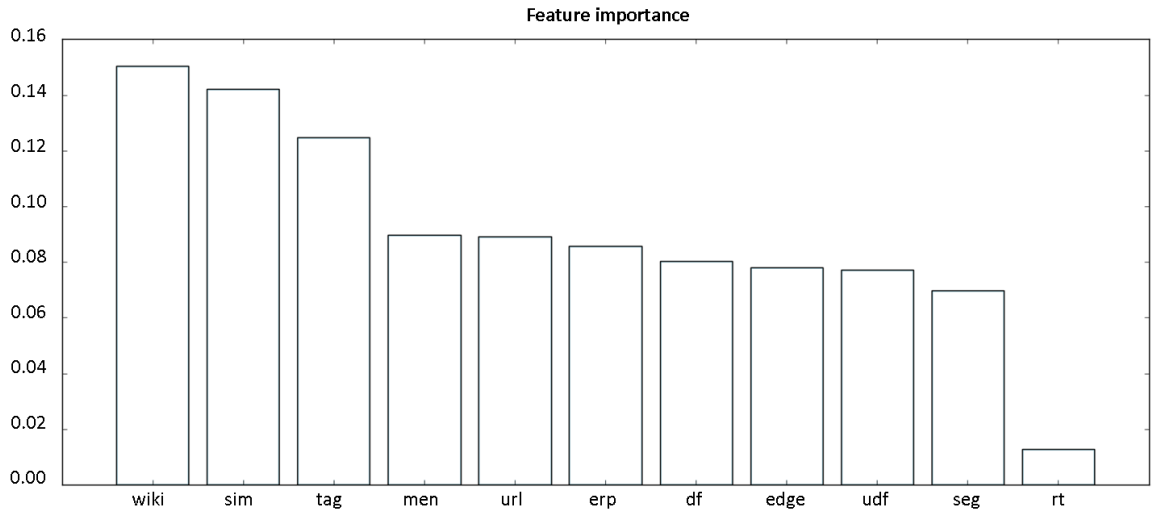

Figure 7: Feature importance.

Model Tuning, Training and Testing

In order to train the model, first a set of candidate events was computed. The resulting 1,427 candidate events were then manually labeled as *1* when referring to a real-world newsworthy event or *0* otherwise. From these, 84 were considered to refer to real events (i.e. labeled as *1*). In order to try to avoid the possibility of overfitting, only 420 (336 randomly chosen samples for the *0* class and the 84 samples for the *1* class) of these 1,427 samples were used. This represents a tradeoff between losing information on one hand, by discarding samples, and trying to increase the quality of the training data, by providing a better balance regarding the number of samples representing each of the classes in that data, on the other.

Grid-search along with cross validation (the GridSearchCV class from Scikit-Learn) was then used in order to find the combination of the hyper-parameters of the SVM model, namely, kernel, C and γ, yielding the best classification results. The best combination found for the values of these parameters is listed in Table 4. 80% of the training data were used for this purpose and cross validation was performed using 5 folds.

This model was then tested with the remaining 20% of the training data, obtaining a precision of 92% and a recall of 65% for class *1* (representing the real events). For comparison purposes, a Random Forest model, comprising 10,000 decision tree classifiers, was also trained and tested on the same data, obtaining a precision of 80% and a recall of 71% on the same class. These results are presented in Table 5.

The final model was created using the best combination of hyper-parameters found and trained using all the training data. This model was then pickled into a file so that it could be used later on, to perform the filtering step. The Python code snippet used to tune, train and test the model is presented in Appendix A, Listing 18.

Table 4: Grid-search results.

Best hyper-parameter values	'C': 100.0, 'gamma': 0.01, 'kernel': 'rbf'

Table 5: Test results for the SVM and Random Forest models.

Model	Classes	Precision	Recall	F1-score	Support
SVM	**0**	0.92	0.99	0.95	67
	1	**0.92**	**0.65**	0.76	17
	Avg/Total	**0.92**	**0.92**	**0.91**	**84**
Random Forest	**0**	0.93	0.96	0.94	67
	1	0.80	0.71	0.75	17
	Avg/Total	**0.90**	**0.90**	**0.90**	**84**

4.4 Precomputed Values Infrastructure

This section discusses the implementation and setup of the Precomputed values Infrastructure block, see Figure 2, used to manage the precomputed values required by the system in order to perform its task. These values were alluded to during the discussion of this block in Subsection 3.6.3 and mapped to the implementation in Section 4.2, namely, the segment probability and the Wikipedia anchor probability.

Upon this discussion a third precomputed value, required due to specific aspects of the implementation of the base system, was also introduced, specifically the segment frequency probability. The section starts by presenting the rationale of the choice of Redis as the storage backend for these values. Next, the process used to compute each of these values is discussed. Only n-grams up to order 3 (i.e. $n \leq 3$) were considered

4.4.1 Storage

As already mentioned, the decision to precompute the values identified as required by the system and just mentioned in the introduction, as to do with performance reasons (i.e. to avoid having to recompute them) and with some specificities of the implementation of the base system. Therefore these values had to be precomputed and persisted in a suitable way so that they could be conveniently looked up later on.

While an in-memory database was not strictly necessary, Redis proved to provide all the necessary features out of the box: easy installation and configuration in Ubuntu 16.04 LTS; simple and straightforward integration with Python (thru the redis package); good documentation; support for collection data types, which proved quite handy during the precomputation phase; fast key look-up and data persistence. When fully loaded with all the precomputed values, the Redis instance consumed approximately 988 Megabytes in memory. The process used to precompute each of these values is detailed next.

4.4.2 Segment Probability

The probabilities for segments, denoted by *P(.)* in Equation 8 (*Pr(.)* in Equation 2 in the original paper) are used during the tweet segmentation phase in order to try to obtain semantically meaningful segments. The original system used the Microsoft Web N-Gram service [104] to lookup these values. Although some online services do provide such capabilities they do not include support for the portuguese language. This means that an infrastructure providing these lookup capabilities had to be implemented. This implementation in turn involved computing these probabilities and then storing them in a suitable form for easy lookup.

More specifically, a period of collected tweets spanning several months was processed in order to extract all n-grams up to order 3 (i.e. $n \leq 3$) and compute their frequencies or counts denoted by $C(w_1 \ldots w_n)$. The probability of each n-gram was then estimated using the Maximum Likelihood Estimate (MLE) [50] as shown in Equation 12, where *N* denotes the total number of words found in the set of tweets used.

It should be noted that this estimation method can present some issues such as the fact that previously unseen n-grams will be assigned zero probability. Several techniques exist to try to correct this issue such as Laplace's Law often referred to as *adding one*, Held out estimation and Good-Turing estimation [50]. This was not a concern in this case as the period chosen to compute these estimates comprised the period used to test the implementation of the framework later on.

$$P_{MLE}(w_1 \ldots w_n) = \frac{C(w_1 \ldots w_n)}{N} \qquad (12)$$

In terms of storage, the first attempt to persist these probabilities was to use serialized Python dictionaries that could then be easily loaded into memory. This option however turned out to be unfeasible due to the large amount of n-grams computed. Instead, a Redis database instance was used for the reasons already mentioned. In total 9,152,120 n-grams, along with their precomputed probabilities were stored in this Redis database instance. The python script used to compute these probabilities is presented in Listing 3.

```
PYTHONPATH=~/Desktop/Dissertation python3 computeNGramCounts.py -d 'dumpdb' -c 'dump' -w '2016-07-01T00:00:00' -u '2016-10-01T00:00:00'
```

Listing 3: Python shell script used to compute the probabilities of segments.

4.4.3 Wikipedia Anchor Probability

The Wikipedia anchor probability, denoted by *Q(s)* in Equation 11 (and also in Equation 3 in the original paper), is used both during the tweet segmentation and the event segment detection phases. In order to compute these probabilities the latest portuguese Wikipedia dump (i.e. ptwiki-latest-pages-articles.xml.bz2[1]) was downloaded from [105]. The uncompressed XML dump file measuring approximately 6.6 GB in size was then parsed.

Only text contained inside anchor blocks encoded as [[…]] in the dump file and excluding images was considered as a potential anchor text candidate. All other blocks such as references, quotations and comments were ignored. Redirect, disambiguation and *Wikipedia:* alike titled pages (e.g. Wikipédia: Log de Uploads) were also left out and not processed.

It should be noted however that using Wikipedia as a well-kept and high quality source presents its own issues. On one hand Wikipedia often addresses trivial subject matters such as

[1] Corresponding to the dump of the 21st of October of 2017.

domestic animals (e.g. dog, cat), hardly of any informational value for the event detection process. On the other hand some anchor blocks often present more than one option for their designation (e.g. [[*Municipal de Santana*|*Estádio Municipal de Santana*]]). Some of these differ slightly in form but can have very different informational values. In the example just given, *Estádio Municipal de Santana* seems clearly more informative than just *Municipal de Santana* as it alludes explicitly to the fact that a stadium (*Estádio*) is being referred to.

An even more extreme example of this, concerns the use of acronyms (or even the real designations) for example, to refer to metric measures (e.g. [[*mm*|*milímetro*]] standing for milimeters), political parties and all kinds of institutions. This presents an obvious problem as the informal nature of twitter is prone to the use of abbreviations, as presented in the analysis discussed in Section 5.2.3.

In fact according to this analysis *mm* is the most used abbreviation found in tweets and seems unlikely to be referring to millimeters all the time (in fact it can have many possible meanings). This also means that perfectly legal acronyms, designations or even names, could be easily misinterpreted as referring to something else (e.g. *SEI* and *PRA* can denote the name of a company or the acronym of an institution, but can also denote a form of the verb *know* (*saber*) and a commonly used abbreviated form of the preposition *to* (*para*) respectively. *Fiz* can denote the name of a person, but is also the past form of the verb *do* (*fazer*) and so on).

Some of these issues were addressed by always choosing the longest choice available for the designation of the anchor, that is to say, the one with the greatest number of words, when more than one option is available. Figure 8 depicts two examples of this, where when faced with more than one option to designate the anchor, its longest form is always chosen. This was done with the intuition that a longer text tends to be more descriptive. This also favors the longer forms of the acronyms, abbreviations, names or designations.

Finally only n-grams appearing more than once as anchors, were considered. This was done with the expectation that more trivial topics and therefore of minor interest, are less likely to be used as anchors. This choice may obviously have an impact in the results of the system, as some potentially newsworthy segments will not be boosted and therefore drop in the rank, increasing the change of their disposal by the system and excluding them from further processing. In total 1,154,330 anchor designations were persisted. Only the n-grams considered were given a probability, all other n-grams are implicitly given zero probability. The python script used to compute these probabilities is presented in Listing 4.

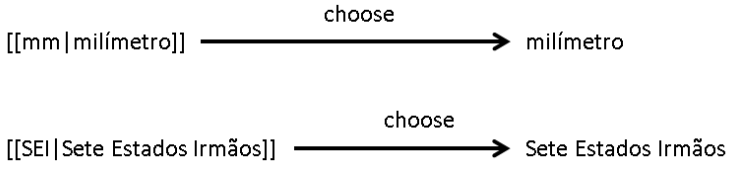

Figure 8: Strategy used for anchor choice,

```
PYTHONPATH=~/Desktop/Dissertation python3 processWikipedia.py -d 'dumpdb' -c 'dump' -i '/home/developer/Desktop/ptwiki-latest-pages-articles.xml' -b 50000 -v 1
```

Listing 4: Python shell script used to compute the anchor probabilities of segments.

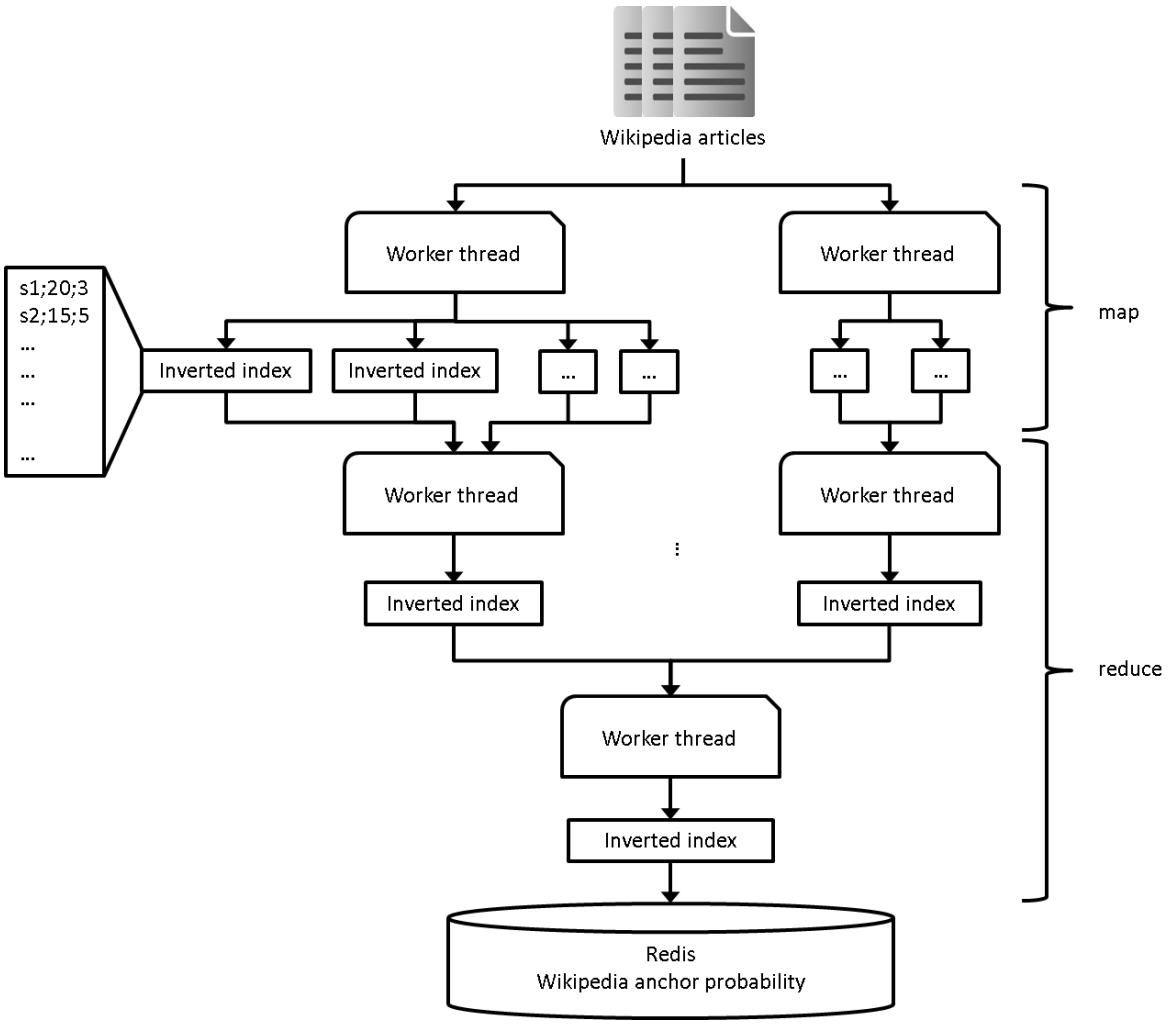

Figure 9: MapReduce procedure used to compute the Wikipedia anchor probabilities for the segments.

To conclude the discussion, a few remarks concerning the actual pipeline implemented to compute these probabilities should be noted. As it proved unfeasible to compute this amount of data purely using memory, a two phased strategy similar to MapReduce had to be implemented. As depicted in Figure 9, the first phase of the procedure consisted in processing the articles into

batches of size 50,000, to create an inverted index for each of these batches. Each entry in the index corresponds to a tuple of the form (*segment*; *article-counts*; *anchor-counts*).

These inverted indices were then merged in pairs until only one remained, during the second phase of the process. The entries found in this last index were then persisted into a Redis database instance in the form (*segment*, *anchor-counts / article-counts*).

4.4.4 Segment Frequency Probability

The probabilities of the frequencies of the segments, denoted by p_s and computed as shown in Equation 13, are used in the event segment detection phase, to detect bursty segments and are specific to the implementation of the base system. This formulation was not formally defined in the original paper, but presented in [11] instead. In the equation depicted, N_t denotes the number of tweets created within time window t, $f_{s,t}$ denotes the frequency of segment s within t (i.e. the number of tweets created in t that contain s) and L denotes the number of time windows t containing segment s.

$$p_s = \frac{1}{L} * \sum_{t=1}^{L} \frac{f_{s,t}}{N_t} \quad (13)$$

Similarly to the process used to compute the probabilities for the segments, presented in Section 4.4.2, the same period of tweets was used to compute these frequency probabilities, which were then persisted in a Redis database instance. This process consisted of two phases: first the $f_{s,ti}$ / N_{ti} ratio was computed for each segment, for each of its time windows and stored in a list in the form (s: [$f_{s,t1}$ / N_{t1}, $f_{s,t2}$ / N_{t2}, …, $f_{s,tL}$ / N_{tL}]). At the end of the process the values in the lists were used to compute Equation 13 for each segment and store the resulting p_s value. In total 1,016,452 n-grams were stored. The python script used to compute these probabilities is presented in Listing 5.

```
PYTHONPATH=~/Desktop/Dissertation python3 computePs.py -d 'dumpdb' -c 'dump' -w '2016-07-01T00:00:00' -u '2016-10-01T00:00:00' -v 1
```

Listing 5: Python shell script used to compute the frequency probabilities of the segments.

4.5 Parameterization

In terms of parameterization, the implemented system requires four parameters, listed in Table 6. Two of these parameters, namely S_t and K were already presented in Section 3.7. Regarding the remaining two parameters, their usage is as follows. The k parameter is used in order to parameterize the Jarvis-Patrick clustering algorithm, implemented to compute the candidate events and used by the Event Segment Clustering component.

The S_m parameter is used to specify the size of the sub-time windows, during the pairwise similarity computation of the event segments, performed also by the Event Segment Clustering

component. This is done in order to try to better distinguish events by taking into account their temporal frequency patterns. This aspect is explained in more detail in Section 3.3 in the original paper or alternatively in Appendix D.

Table 6: Parameters used to parameterize the implemented framework.

Parameter	Description
S_t	The size of time window *t*
K	The top-K tweet segments to retain
k	The k-nearest neighbors to consider when deciding if two segments should be clustered together (i.e. both must appear in each other's k-nearest neighbors)
S_m	The size of sub-time window *t'*

Summary

This chapter presented the implementation details of two of the blocks of the system, namely the Event detection pipeline and the Precomputed values Infrastructure. The several components composing these blocks were discussed in terms of their respective implementations, mapping the implementation choices to the requirements previously identified. The implementation of the segmentation algorithm and of the Jarvis-Patrick clustering algorithm variant, the presentation and empirical validation of the change proposed in the weighting scheme used to rank the segments and the tuning, training and testing of the SVM model used to perform the filtering step, are some of the highlights discussed during the presentation of this implementation.

The difficulties encountered during the implementation of the infrastructure used to support the precomputed values required by the system, as well as the technical solutions employed in order to overcome those difficulties, were also a topic of the discussion. The several parameters required by the system were presented at the end of the chapter. The third block of the system, namely the Data source Infrastructure, was not addressed and is presented in the next chapter.

5 Data Preparation, Cleaning and Storage

This chapter discusses several aspects related to the dataset. The chapter starts with a brief introduction to the dataset itself. The exploratory data analysis performed is presented next. For this purpose several metrics were computed and are discussed. This is done in order to gain a deeper insight about the data, so that this knowledge can be leveraged to make better and more informed decisions, later on. Then the implementation of the Data source Infrastructure block, presented in Subsection 3.6.2, see Figure 2 in Section 4.2, is detailed. The pre-processing of the data, as well as the rationale behind the choice concerning the database backend used to persist the dataset, are the main topics of this discussion.

5.1 General Description

The dataset used consists of a set of tweets collected from the Twitter Search API [106] by a JAVA agent and stored in a Cassandra database as part of TVPulse [53], a project developed at Instituto de Telecomunicações de Aveiro, to detect TV highlights in Social Networks. Only tweets created in Portugal are collected, as the focus of research of the aforementioned project is the portuguese community. Given this, and mostly due to time constraints, it was then decided to use this existing dataset as opposed to purposely collect and build a dataset to use in this work. This choice may present some downsides, some of which will be addressed later on, as the data collected as well as the process collecting this data, are beyond the control of this work and its author.

5.2 Dataset Analysis

This section presents the several metrics collected during the exploratory data analysis conducted on the dataset. The goal of this analysis, as already mentioned, is to obtain a better understanding of the data, so that more informed decisions and or tradeoffs can be made, later on. In order to do this, several metrics related to the tweets and users distributions were computed and are discussed first. Finally some of the features related to the tweets themselves (i.e. concerning the text of the tweets) and considered relevant, are analyzed.

5.2.1 Tweets Distribution

The collected tweets span four years, from 2014 to 2017, with most of the tweets distributed over 2015 (16,550,792 tweets collected representing 51.9% of the total of collected tweets) and 2016 (12,566,003 tweets collected representing 39.4% of the total of collected tweets). Representing a mere 8.6% in total, the years 2014 and 2017 corresponding to 5.3% and 3.3% respectively, were therefore considered not representative enough to be used in the remainder of the task.

These results are depicted in Figure 10. The asymmetry found in the number of tweets collected per year, as to do with some aspects related to the process of collecting these tweets, and as already stated, is beyond the control of this work.

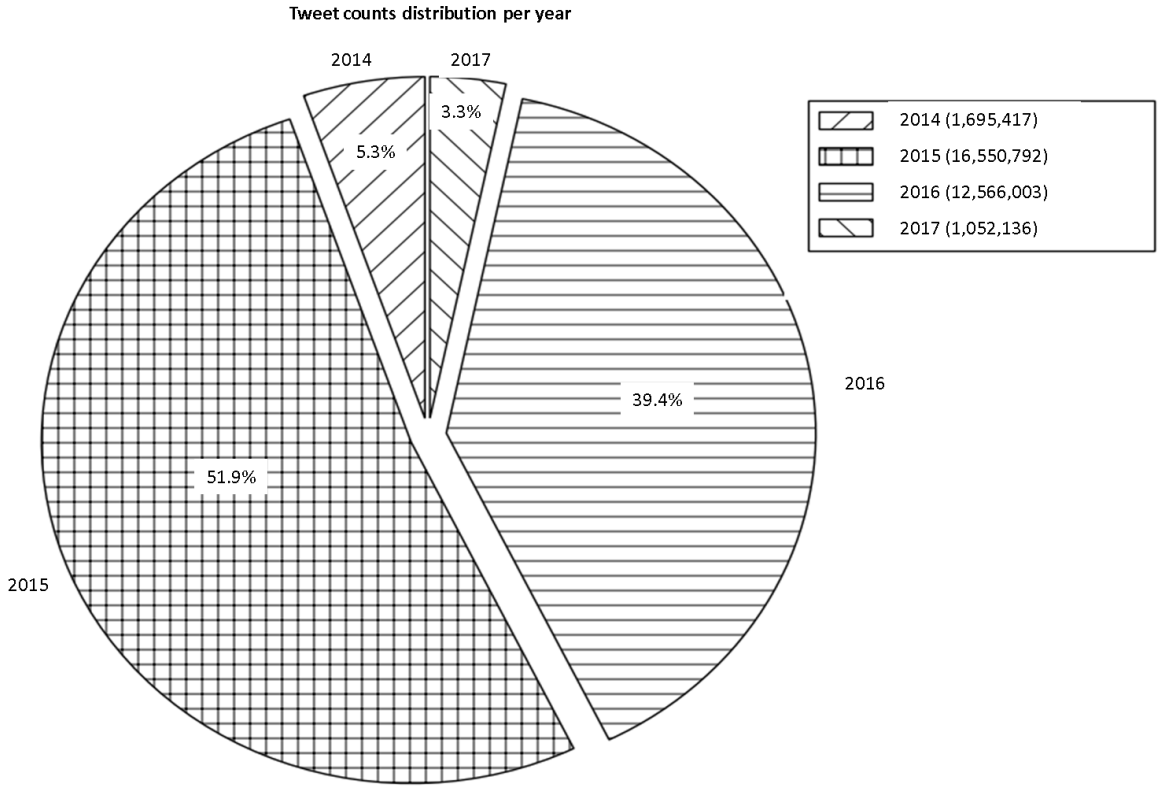

Figure 10: Distribution of the number of tweets collected per year.

An analysis over the distribution of the tweets collected per month, was also performed and is shown in Figure 11. For completeness, the years 2014 and 2017 were also included. From the analysis of this figure, it can be concluded that for the year 2015, all months have collected tweets, with a clear drop in the collecting process during August and also to a lesser extent, February, March and December. For the year of 2016 no tweets were collected during November and December, with a clear drop in May.

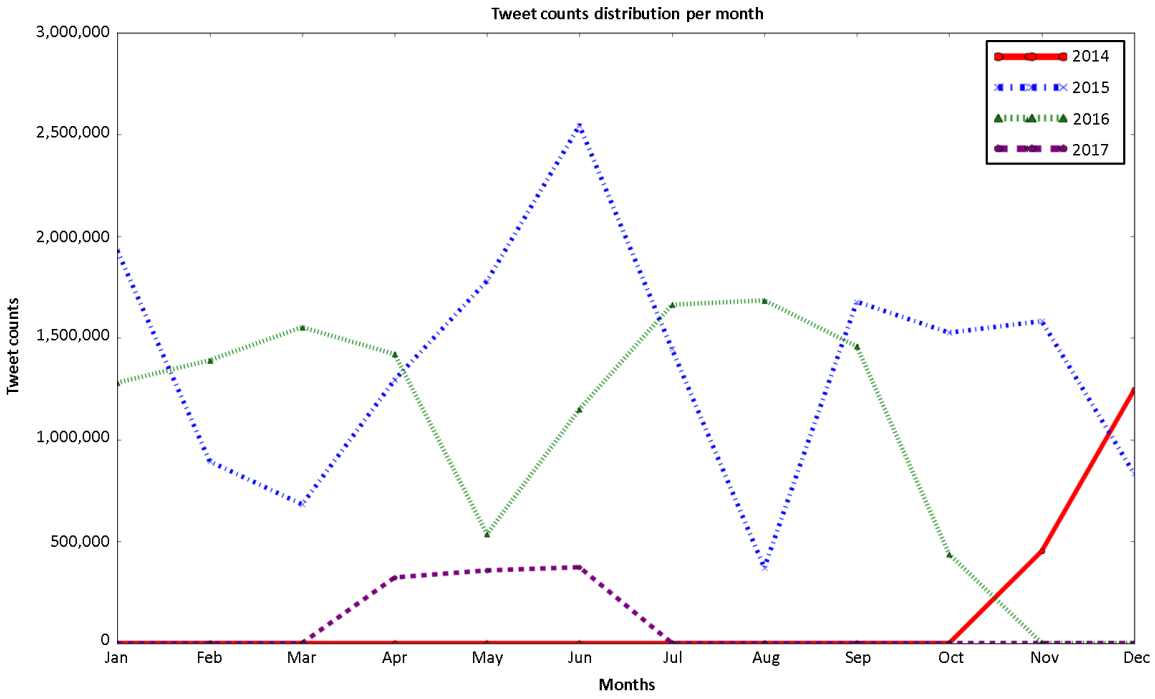

Figure 11: Distribution of the number of tweets collected per month for each year.

Finally, an analysis over the distribution of the collected tweets per day was also performed. The results are presented by trimester in Figure 12, Figure 13, Figure 14 and Figure 15 respectively. Again and for completeness, all the four years are included in the visualizations. These results confirm the conclusions already mentioned during the monthly analysis, with the added benefit of allowing for a more fine grained insight over the true representativeness of each month in terms of collected data.

As an example of this let us consider January of 2015. At first sight, this month should present no issues, being the second month with more collected data for this particular year (almost 2 million tweets collected and only surpassed by June), see Figure 11. However, a closer look at Figure 12 indicates that this month lacks data for a consecutive period of 8 days (from the 19th to the 26th inclusive).

Depending on the task being performed (e.g. the analysis of trends during that particular month), this lack of data can greatly impact the meaningfulness and the generalization power of any conclusions drawn from the analysis of the tweets collected over this period (i.e. January of 2015). Again, depending on the specific task to be performed, this may turn out to be relevant or not. It is however important to be aware of such a potential pitfall, so that proper strategies can be devised to deal with these periods with missing data if needed.

For the purposes of this work this analysis makes it possible to identify the most consistent periods in terms of the collected tweets. For the year 2015, these would be from the 7th of May to the 30th of June and from the 5th of October to the 30th of November. For the year 2016, these

would be from the 13th of January to the 31st of March and from the 1st of July to the 30th of September.

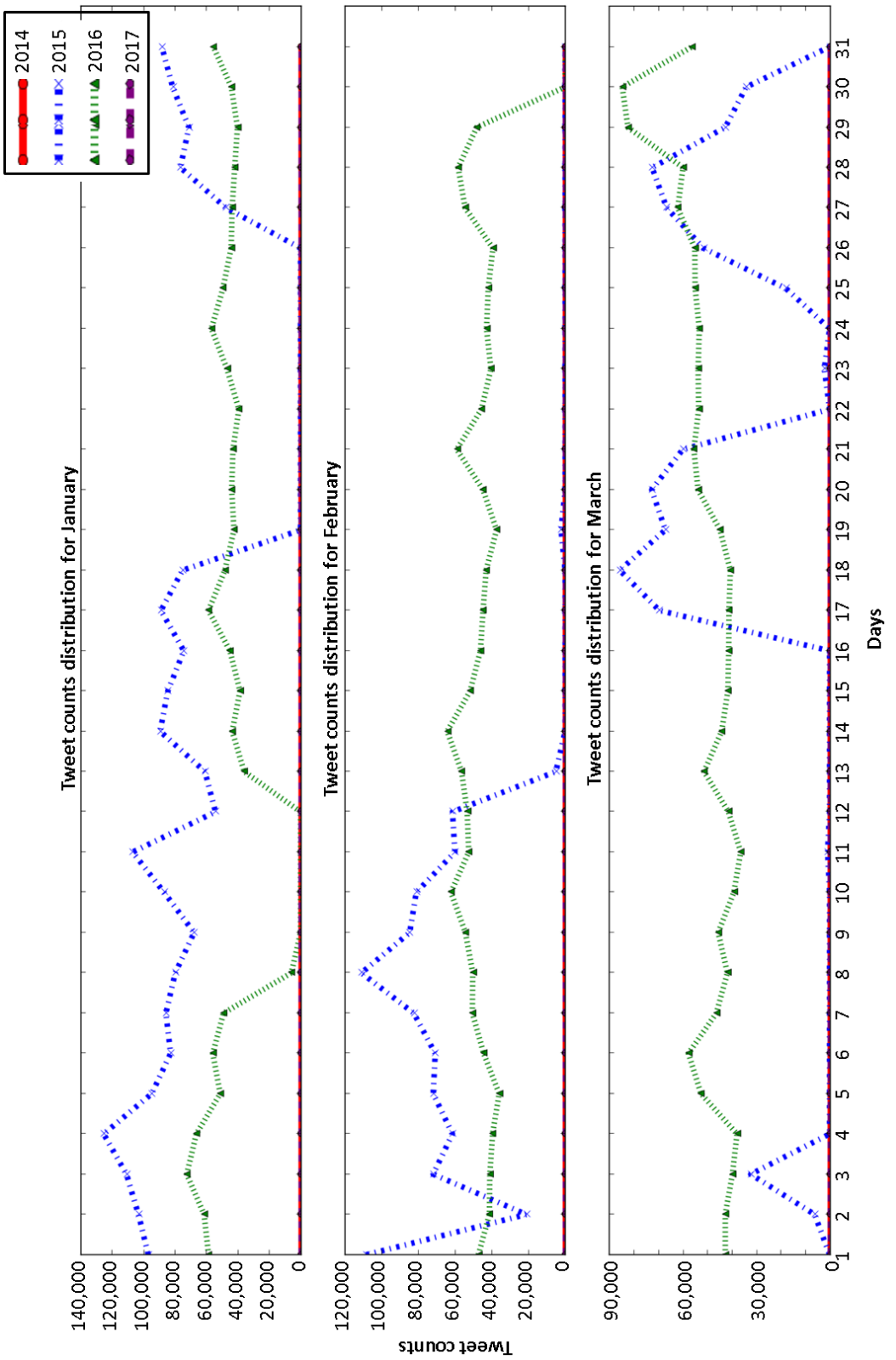

Figure 12: Distribution of the number of tweets collected per day for the first trimester.

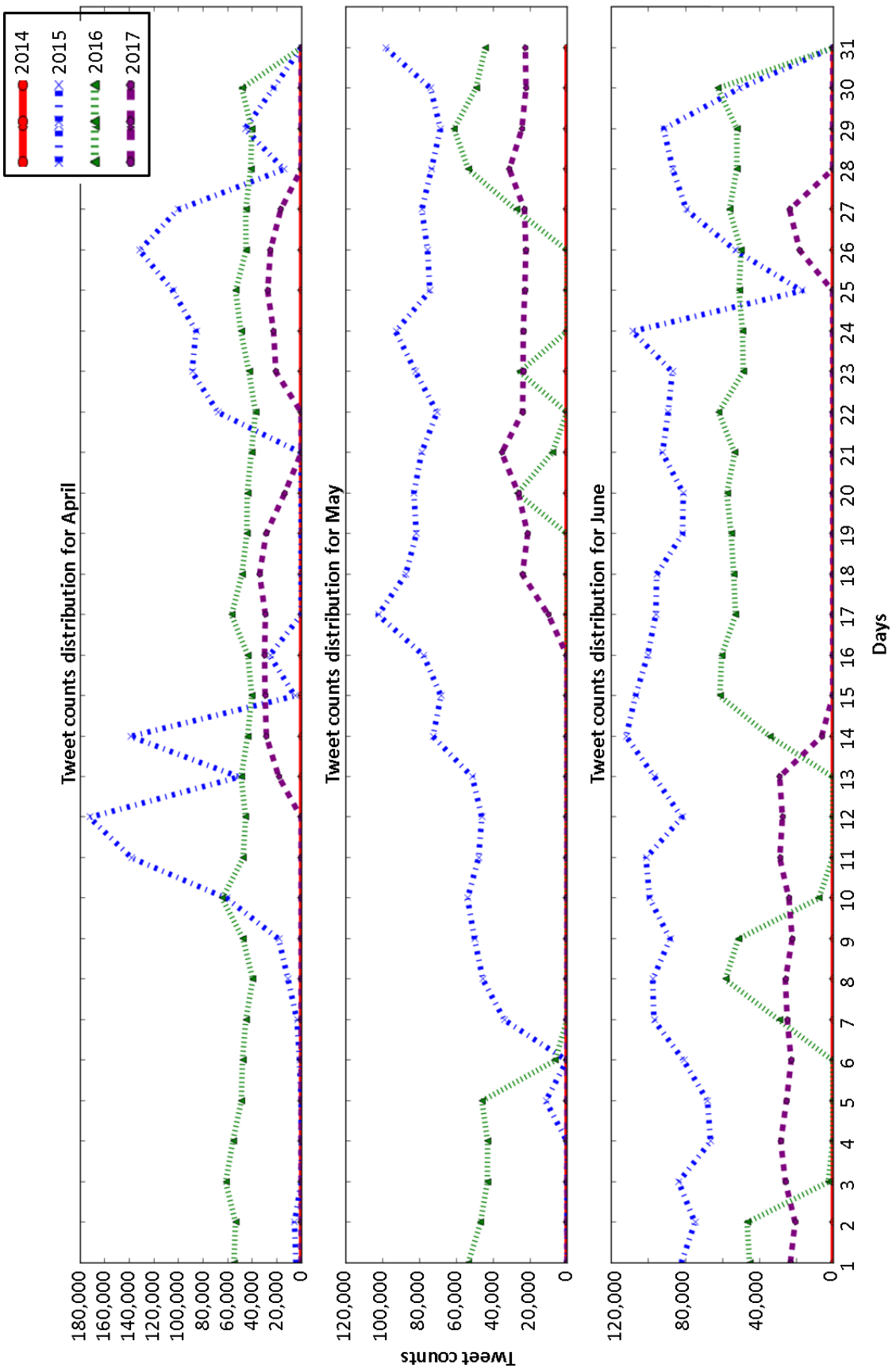

Figure 13: Distribution of the number of tweets collected per day for the second trimester.

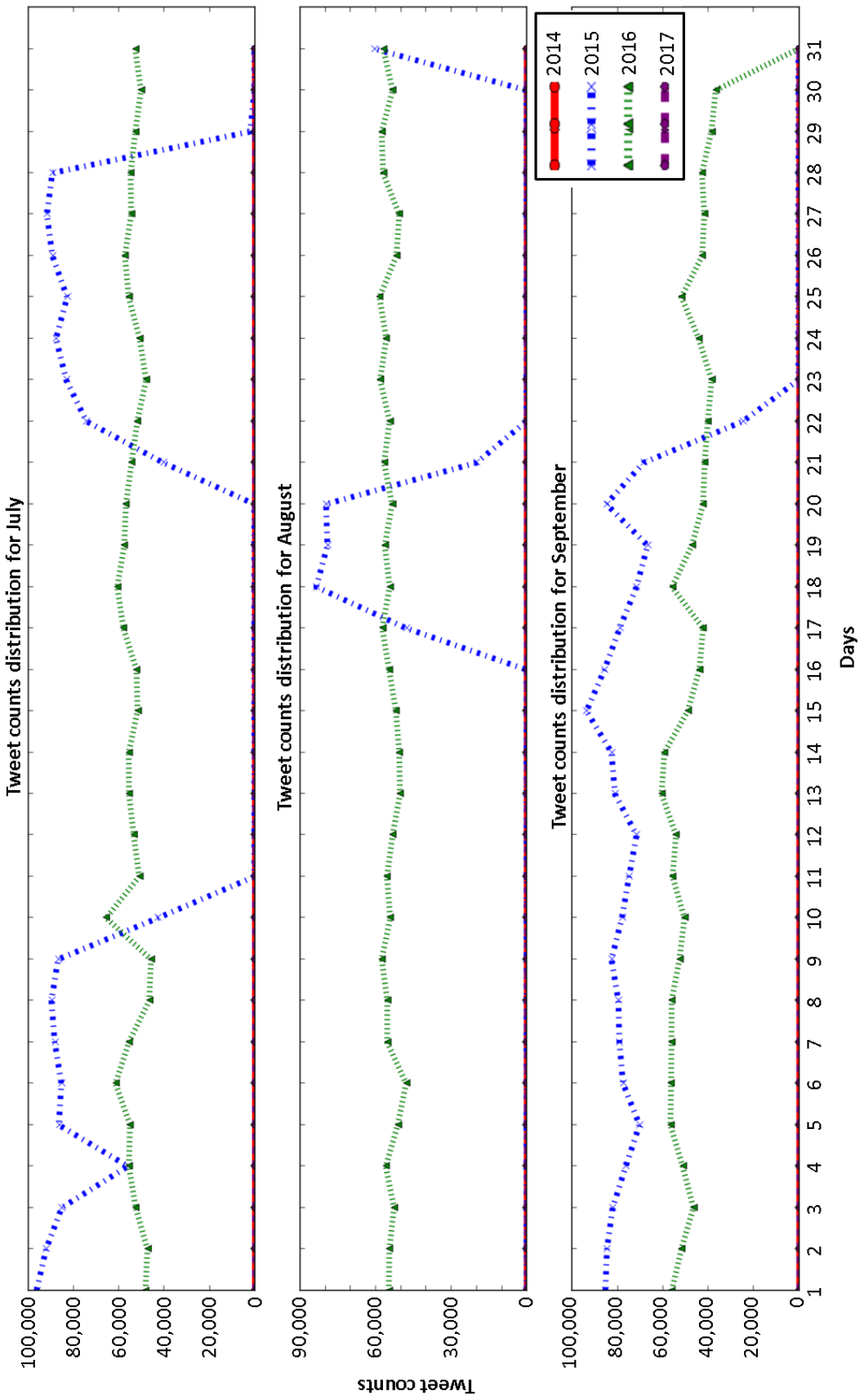

Figure 14: Distribution of the number of tweets collected per day for the third trimester.

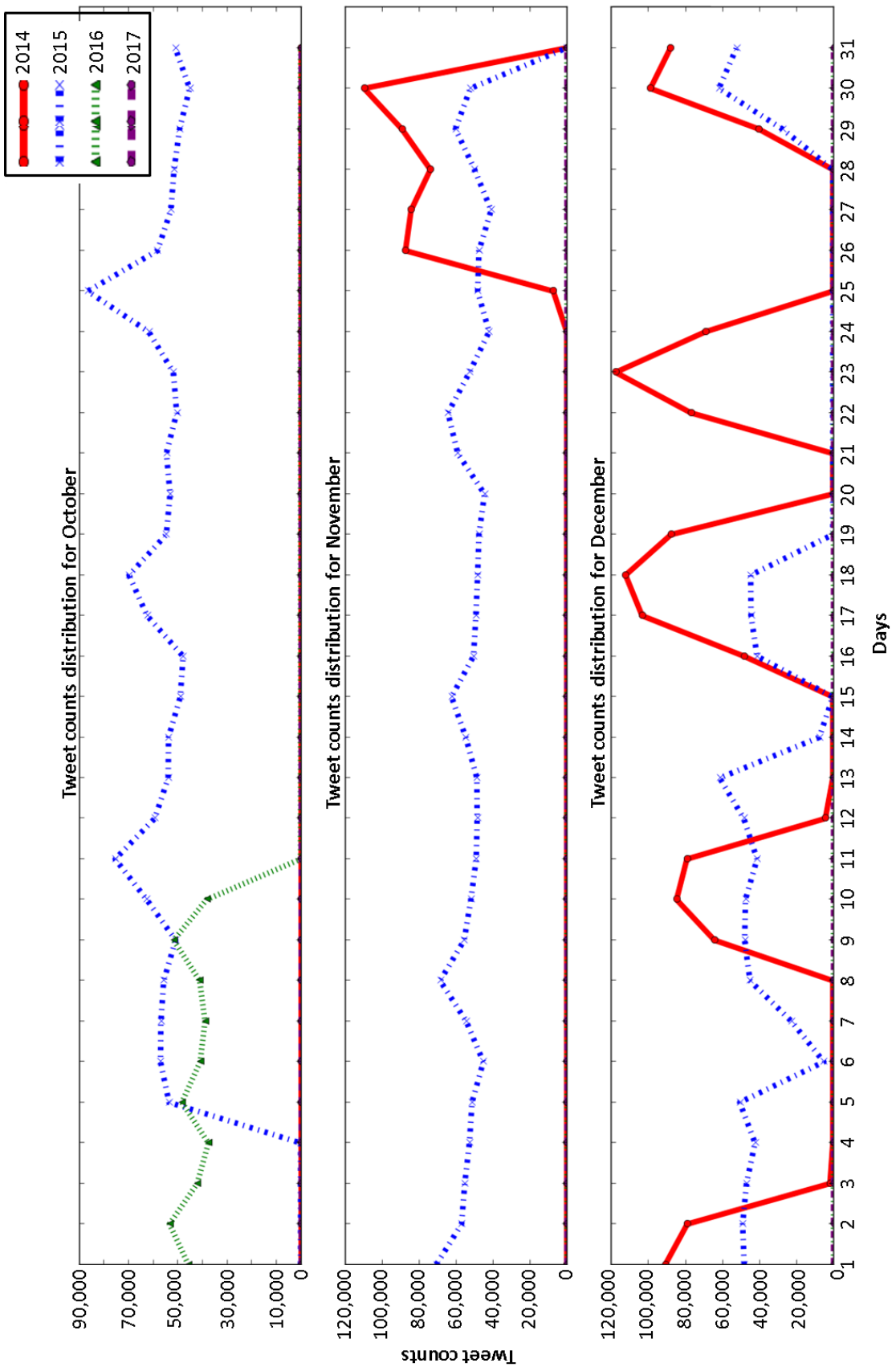

Figure 15: Distribution of the number of tweets collected per day for the fourth trimester.

As another example of the different levels of analysis granularity, regarding the information contained in the dataset and the goodness of information that can be obtained from them, let us consider the tweets collected over each day, per hour. Figure 16 depicts the number of tweets collected for each day, per hour, during March of 2016 (it should be noted that all days in this month have collected data, refer to Figure 12).

Immediately a distinctive pattern seems to emerge, characterized by relatively well defined and distinguishable peeks of activity, such as the ones that seem to converge to a maximum roughly between 11 am and 12 pm, and 21 pm, respectively and the one that seems to converge to a minimum, occurring roughly between 4 am and 5 am. This kind of analysis is not the objective of this work and such conclusions would of course have to be more formally proven. The point to be taken is that a thorough analysis of the dataset can convey very useful information and can ultimately, as in this case, provide very insightful hints about possible patterns lurking inside the data.

It should be noted however, that if a more systematic analysis over the whole dataset were to prove this trend of maximum and lower activity periods to be true, then periods with a really low rate of activity (e.g. less than 500 tweets created) could potentially be discarded with no significant loss of information, while also improving computational, storage and time costs regarding the persistence and processing of the tweets (i.e. less tweets to store and process). For the purposes of this work no such analysis was performed and all periods were taken into account.

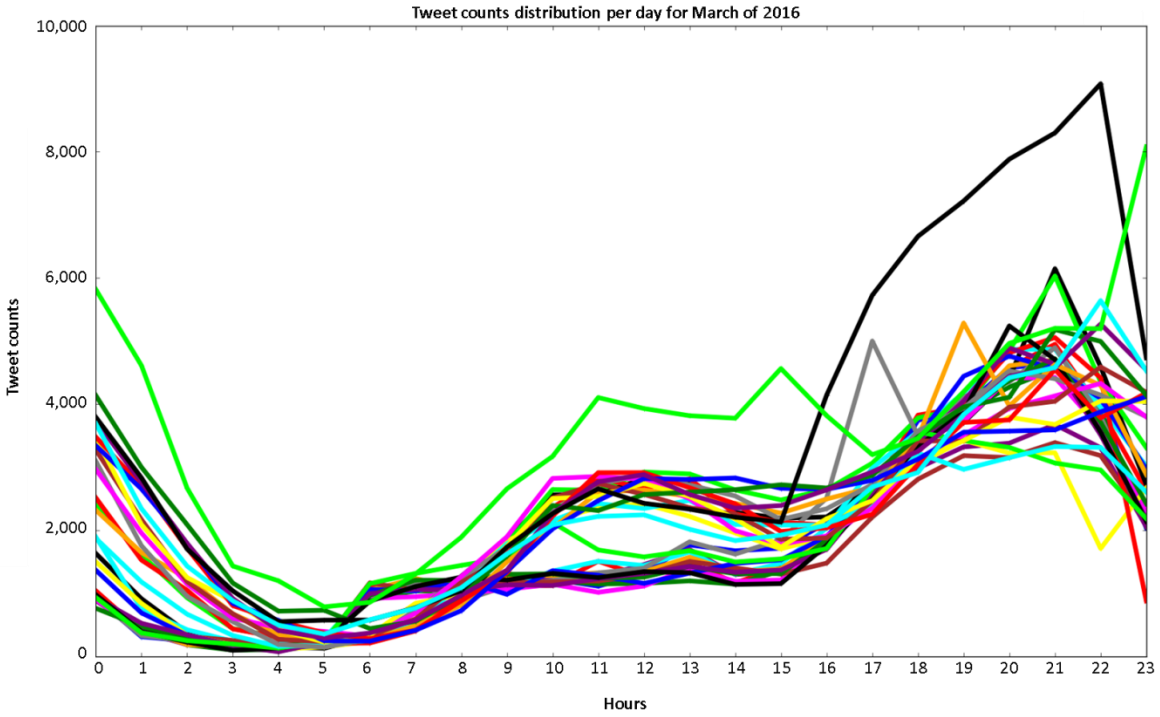

Figure 16: Distribution of the number of tweets collected per hour (for each day) during March of 2016.

5.2.2 Users Distribution

The information concerning the users was not expected to be of much importance for the proposed task, at least at this point, therefore the analysis performed was not as detailed. Nevertheless, a brief analysis to try to characterize user activity (i.e. the number of tweets created by each user) is presented in Figure 17 and Figure 18.

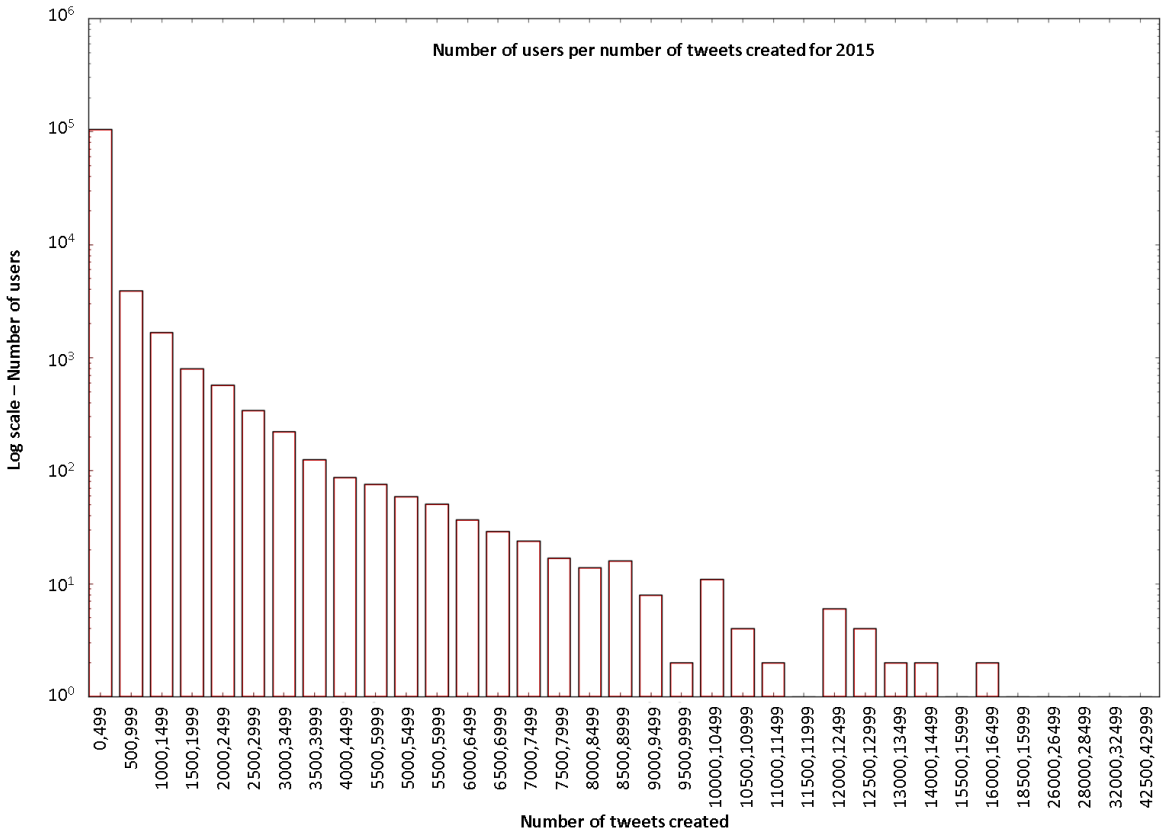

Figure 17: The number of tweets created during 2015 (x axis) and the respective number of users (y axis).

Table 7: Number of unique users per year.

Year	Number of unique users
2014	24,751
2015	112,626
2016	121,323
2017	35,919
Total	**217,398**

For ease of visualization it was decided to group the number of tweets created (the x axis) into bins of size 500, so for example, the leftmost bar in each of the graphs (i.e. the first bar) represents the number of users (the y axis) that created between 0 and 499 tweets inclusive, for the

specified year (the fact that the bins start with 0 was an implementation detail that was not corrected in the meanwhile). The number of unique users per year is also shown in Table 7.

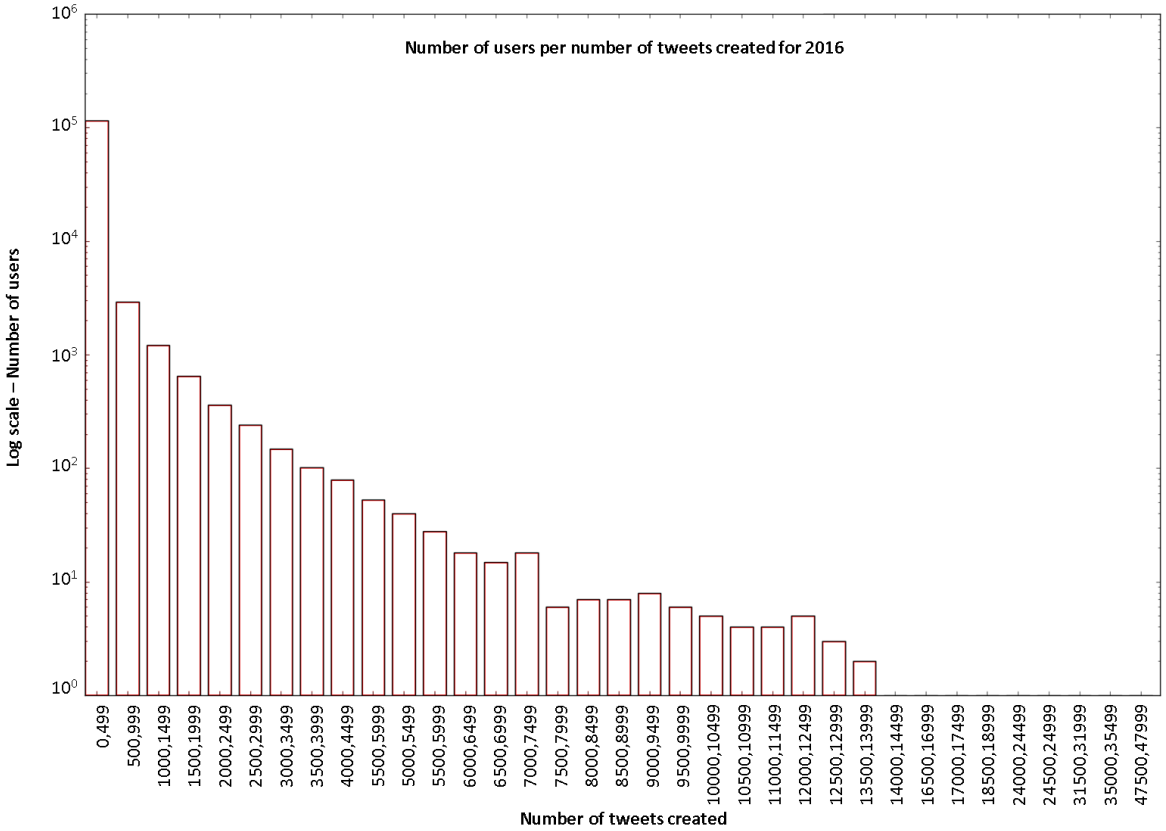

Figure 18: The number of tweets created during 2016 (x axis) and the respective number of users (y axis).

As shown in Figure 17 and Figure 18, both years 2015 and 2016, seem to follow an identical pattern. From this observation, two facts seem to be noteworthy:

- the huge number of users, approximately 10^5, that have created less than 500 tweets over the course of a whole year. In this case it could be interesting to differentiate between a user who seldom tweets and a user who tweets only at some specific circumstances. Such information could then be used to decide if such tweets should be discarded from further analysis or not. Of course, these numbers may not reflect reality as the process collecting the tweets is not perfect and does not collect the entirety of the tweets created over time.
- the suspiciously large amounts of tweets created by a few users (the right side of the graph). According to the graph for 2016, for example, the user with the highest number of tweets created, reached almost 50 thousand tweets (47,909 to be more precise, see Listing 6). Such an amount seems clearly suspicious and may be a strong indication of some kind of automated program injecting tweets into the network, the so called bots or TwitterBots and cyborgs [107]. In fact, a simple inspection of some of the tweets created by a few of these users, seems to indicate just that, as clearly the content of the tweets seems to be the

product of some kind of automated process, as opposed to have been created by a human. Some of these examples are listed in Table 8. Again, depending on the task to be performed, these automated tweets may be considered worthless and even introduce unwanted noise. In which case a closer inspection would have to be performed in order to detect these non-human user accounts and proceed to the removal or filtering of their tweets from the data. For the purposes of this work the top ten users in terms of the number of tweets created was manually inspected, for each of the years, in order to identify all non-human users. The tweets created by the non-human users identified, were then removed from the dataset in order to reduce noise.

The script used to compute these and the tweets distribution statistics, is presented in Listing 7.

```
{ "_id" : { "userId" : 1361619691 }, "count" : 47909 }
```

Listing 6: Query result, showing the highest number of tweets created in 2016 along with its respective user id.

Table 8: Sample of some of the tweets created by a few of the users with the highest number of tweets created.

Tweet text
6. Kendall\n7. #LouisWeSupportYou\n8. Happy New Year\n9. Faltam 10\n10. WhatsApp\n\n2015/12/31 23:53 WET #trndnl https://t.co/uLzQlByvJf
1. #VouLevarPara2016\n2. #VoltaAoMundo2016\n3. Dubai\n4. #HarryLouYear\n5. Feliz Ano Novo\n\n2015/12/31 23:53 WET #trndnl https://t.co/uLzQlByvJf
Wind 0,9 km/h NE. Barometer 1025,9 hPa, Rising slowly. Temperature 2,4 °C. Rain today 0,0 mm. Humidity 97%
"WhatsApp' just started trending with 1132599 tweets. More trends at https://t.co/uLzQlByvJf #trndnl

```
PYTHONPATH=~/Desktop/Dissertation python3 datasetStats.py -d 'dumpdb' -c 'dump' -v 1
```

Listing 7: Command shell to compute the statistics related to the tweets and users distributions.

5.2.3 Tweets Content Related Statistics

The text composing the tweets (i.e. the message itself) was expected to contain valuable information, therefore several statistics concerning this aspect were computed. This was done in order to better characterize the dataset in this regard and also as a pre-step in the feature selection process (i.e. identify possible candidates to be used as descriptive features of the content of the tweets). These feature candidates could be for example idiomatic (i.e. native speaker related) or twitter specific (i.e. linguistic artefacts used only in twitter) constructs.

Taking into account the analysis performed earlier in sub subsection 5.2.1, it was decided to only include in this analysis the tweets created in 2015 and 2016, therefore leaving the other two years, 2014 and 2017, out due to their lack of representativeness in the dataset (only 8.6%). More specifically only the year 2016 was used to perform this analysis. This was done because: 1) although having less tweets collected, the collection process seems to have been overall more consistent in 2016 when compared to 2015. Also 2016 is the most recent year; 2) including both years would increase both the computational and time costs of the analysis; 3) it is not expectable that these years differ significantly in terms of the statistics to be computed.

The first analysis performed, targeted word usage, specifically the 50 most commonly used words. As depicted in Figure 19 and also in Table 9, most of these words are very short in terms of their length, consisting of verbs (e.g. *ter*, *és*, *vai*, *sei*, *fazer*), words of common use (e.g. *agora*, *tudo*, *hoje*, *dia*, *ainda*), including slang (e.g. *bué*, *tou*) and some swear words (elided from the plot and replaced by the * character), or non-real words (e.g. *mm*, *pq*, *q*, *n*, *c*). Some of these non-real words (e.g. *mm*, *pq*) in turn, stand out as good candidates for abbreviations of common use.

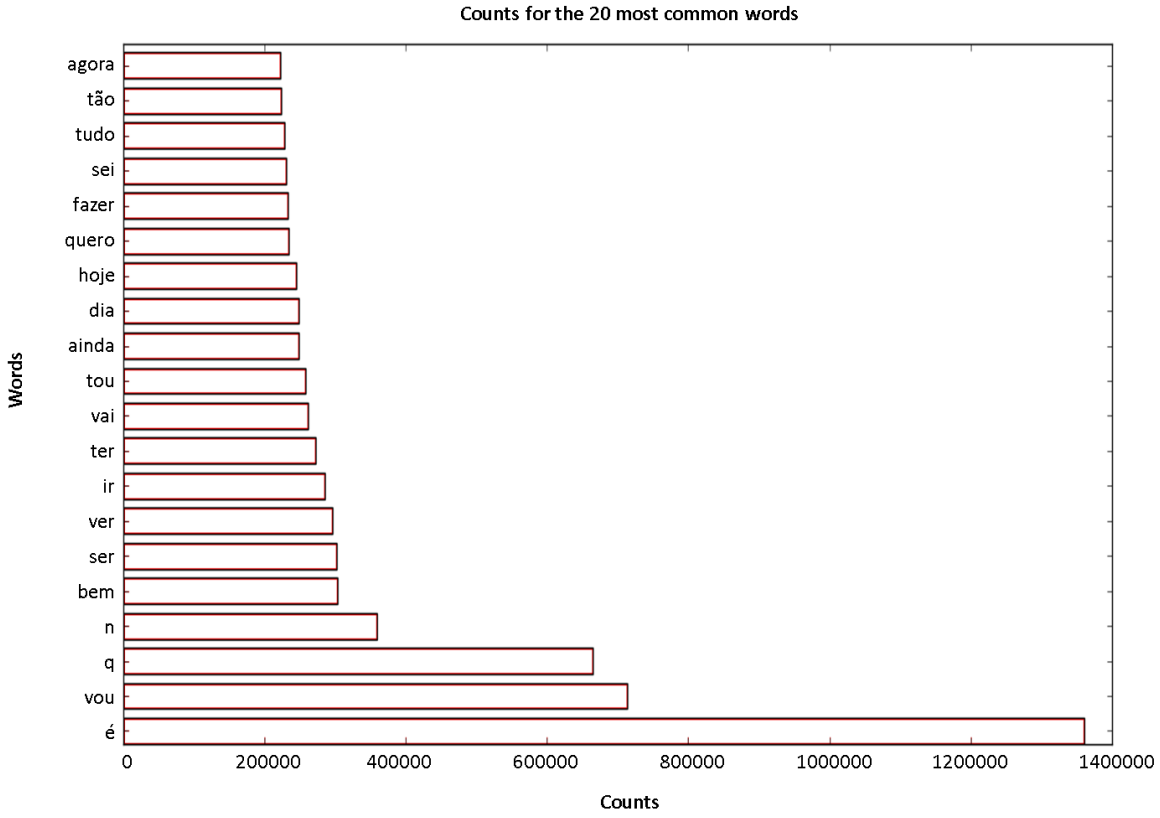

Figure 19: Distribution over the counts of the 20 most common words found in the tweets collected in 2016.

Table 9: List of the 50 most common words.

50 most common words
é, vou, q, n, bem, ser, ver, ir, ter, vai, tou, ainda, dia, hoje, quero, fazer, sei, tudo, tão, agora, p, nada, casa, dormir, nao, sempre, amanhã, portugal, mim, vida, mm, c, *, melhor, fds, , bue, acho, assim, lá, pessoas, bué, és, porque, bom, aqui, pq, …, estar, nunca

One last point to note seems to be the inconsistency of users when following grammatical rules, concerning accentuation for example (e.g. '*nao'* instead of the correct form '*não'*, but '*é´*, *´tão*' and '*amanhã*' in the correct form). Two obvious outcomes seem to follow from this analysis. Firstly, in terms of event detection, these words do not seem to be very informative (with the exception of the word *portugal*) and could therefore be treated as stopwords (i.e. non informative words due to their frequent occurrence and use) and be removed as a way to reduce noise. Secondly, the informal nature of twitter seems to pose additional challenges, such as in the case of misspellings, concerning the proper accentuation in words (although this was already expectable).

For the purpose of this work, these words were not excluded and all words containing accentuations were normalized into their respective non-accentuated common form. This choice however may have an impact in the results, as some similar words with very distinct meanings (e.g. *pais* meaning *parents* and *país* meaning *country*) when properly accentuated, are now treated as a single form, conveying the same meaning. This issue is further explored when discussing the results in Section 6.

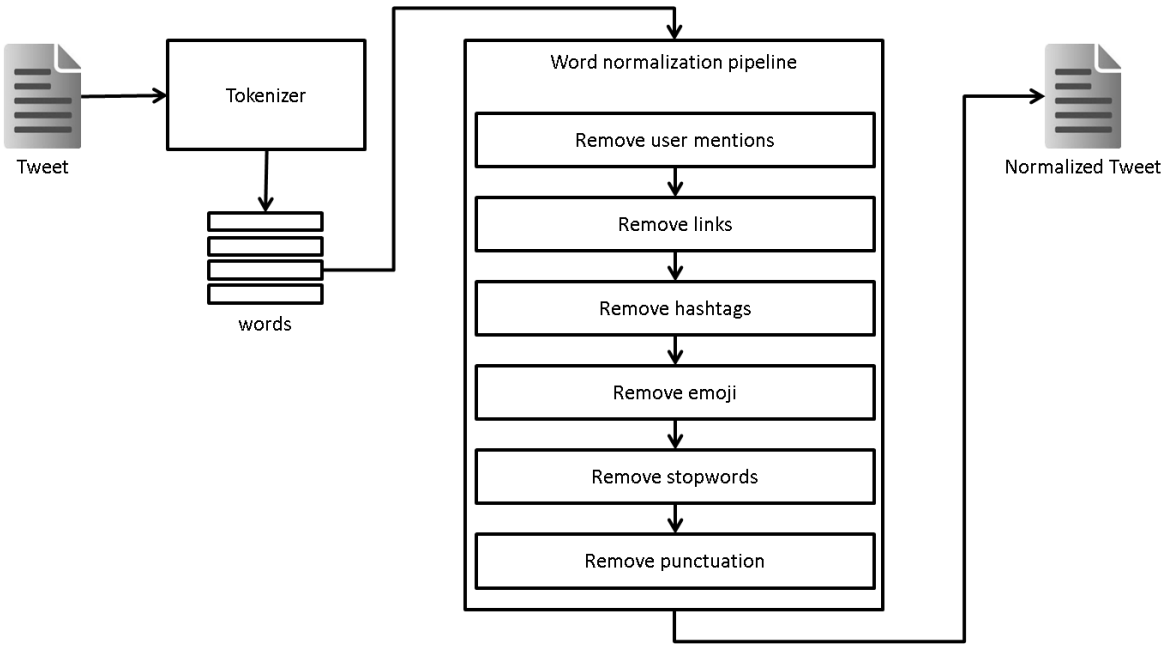

Figure 20: Word normalization pipeline used to compute tweet content related statistics.

As a final aside, it should be noted that in order to compute this and all subsequent analyses, the text was first normalized according to the pipeline depicted in Figure 20. The tweet

content or text was first tokenized into a list of words. From these all user mentions, links, hashtags, emoji, punctuation and stopwords were then removed. This was done, as these do not convey any meaningful information for the purposes of the analysis to be conducted.

Tokenization was performed using TweetTokenizer, a specialized tokenizer for tweets integrated with NLTK. Clearly the list of stopwords used (i.e. the list of stopwords for the portuguese language, integrated with NLTK) may not be sufficient or the most appropriate given the specificity of the written content of these tweets. The full list of stopwords used is presented in Listing 8.

```
['de', 'a', 'o', 'que', 'e', 'do', 'da', 'em', 'um', 'para', 'com', 'não', 'uma', 'os', 'no', 'se', 'na', 'por', 'mais', 'as', 'dos',
'como', 'mas', 'ao', 'ele', 'das', 'à', 'seu', 'sua', 'ou', 'quando', 'muito', 'nos', 'já', 'eu', 'também', 'só', 'pelo', 'pela',
'até', 'isso', 'ela', 'entre', 'depois', 'sem', 'mesmo', 'aos', 'seus', 'quem', 'nas', 'me', 'esse', 'eles', 'você', 'essa',
'num', 'nem', 'suas', 'meu', 'às', 'minha', 'numa', 'pelos', 'elas', 'qual', 'nós', 'lhe', 'deles', 'essas', 'esses', 'pelas',
'este', 'dele', 'tu', 'te', 'vocês', 'vos', 'lhes', 'meus', 'minhas', 'teu', 'tua', 'teus', 'tuas', 'nosso', 'nossa', 'nossos',
'nossas', 'dela', 'delas', 'esta', 'estes', 'estas', 'aquele', 'aquela', 'aqueles', 'aquelas', 'isto', 'aquilo', 'estou', 'está',
'estamos', 'estão', 'estive', 'esteve', 'estivemos', 'estiveram', 'estava', 'estávamos', 'estavam', 'estivera',
'estivéramos', 'esteja', 'estejamos', 'estejam', 'estivesse', 'estivéssemos', 'estivessem', 'estiver', 'estivermos',
'estiverem', 'hei', 'há', 'havemos', 'hão', 'houve', 'houvemos', 'houveram', 'houvera', 'houvéramos', 'haja',
'hajamos', 'hajam', 'houvesse', 'houvéssemos', 'houvessem', 'houver', 'houvermos', 'houverem', 'houverei',
'houverá', 'houveremos', 'houverão', 'houveria', 'houveríamos', 'houveriam', 'sou', 'somos', 'são', 'era', 'éramos',
'eram', 'fui', 'foi', 'fomos', 'foram', 'fora', 'fôramos', 'seja', 'sejamos', 'sejam', 'fosse', 'fôssemos', 'fossem', 'for',
'formos', 'forem', 'serei', 'será', 'seremos', 'serão', 'seria', 'seríamos', 'seriam', 'tenho', 'tem', 'temos', 'tém',
'tinha', 'tínhamos', 'tinham', 'tive', 'teve', 'tivemos', 'tiveram', 'tivera', 'tivéramos', 'tenha', 'tenhamos', 'tenham',
'tivesse', 'tivéssemos', 'tivessem', 'tiver', 'tivermos', 'tiverem', 'terei', 'terá', 'teremos', 'terão', 'teria', 'teríamos',
'teriam']
```

Listing 8: List of stopwords used.

Still concerning word usage, several other analyses were performed regarding more specific aspects, such as the most commonly used abbreviations, long words and repetitions (i.e. words with character repetitions). For completeness, an analysis over the hapaxes, that is, words that occur only once, was also performed.

Concerning abbreviations, depicted in Figure 21 and also in Table 10, it can be seen that seven of these (i.e. *mm*, *fds*, *pq*, *xd*, *mt*, *hj* and one more, elided from the plot and substituted by the * character, as it can be easily related to a common swear word) are clearly more used than the rest. Other than that, it can also be seen that abbreviations are used to convey several different purposes: 1) to mention real products (e.g *pc*); 2) function as emoticons, to denote emotion (e.g. *xd*); 3) abbreviate commonly used words (e.g *hj* instead of *hoje*, *pq* instead of *porque*), including portuguese swear words and well-known english cursing expressions (elided from the plot); 4) for an assortment or other meanings, including some difficult to discern without knowing their specific context of use (e.g. *nng*, *pqp*, *nnc*).

Again, most of these abbreviations do not convey any meaningful information in terms of event detection and could be removed. In the case of mentions to real entities such as products, these abbreviations could be normalized into their longer form (e.g. *computador* instead of *pc*).

However given the informal nature of twitter, this normalization may prove to be difficult for some expressions. As an example of this consider the abbreviation *bb* (not shown in the plot). A brief inspection over some of the tweets containing this abbreviation reveals at least two completely different meanings: *bébé* (baby) and *B.B. King*. Given this ambiguity of usage, the real intended meaning can only be derived by knowing the context, a much harder task.

For the purposes of this work, no abbreviation removal or normalization was performed. As a final remark it should be noted that only words two or three characters long, containing no vowels, punctuation or numbers, were considered as potential abbreviations. This was done to avoid considering real words or expressions such as *ser*, *a*, *o*, *10* and assuming that most abbreviations are no longer than three characters.

Table 10: List of the 50 most common abbreviations.

50 most common abbreviations
mm, fds, pq, xd, *, mt, hj, td, qd, dps, qnd, cmg, dm, km, vcs, *, fdd, gd, msm, nd, pt, smp, cm, sp, tb, qmd, tbm, tt, gnd, tp, fdc, tnh, nnc, rt, pqp, tds, kkk, nng, tmb, qr, msg, *, pc, jr, fg, fzr, nc, pls, mc, np

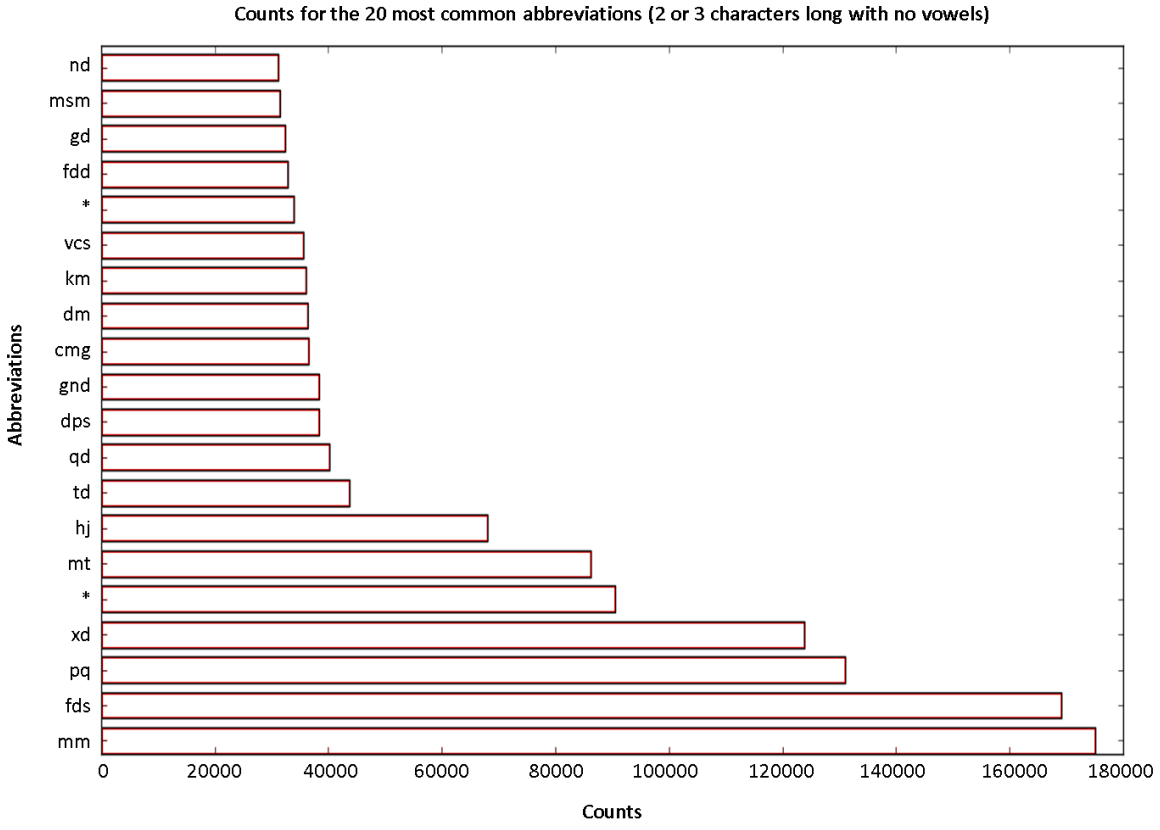

Figure 21: Distribution over the counts of the 20 most common abbreviations found in the tweets collected in 2016.

The analysis concerning the usage of long words, shown in Figure 22 and also in Table 11, shows that as expected, these words are less used (the most common long word *definitivamente* occurs less than 2,500 times) with four of these (i.e. *definitivamente*, *psicologicamente*,

independentemente, *congratulations*) being clearly more used than the rest. Again most of these words are not very informative (e.g. all words ending with the suffix *mente*), although several references to places (e.g. *montemor-o-novo*, *albergaria-a-velha*) and entities (e.g. *primeiro-ministro*) can be found.

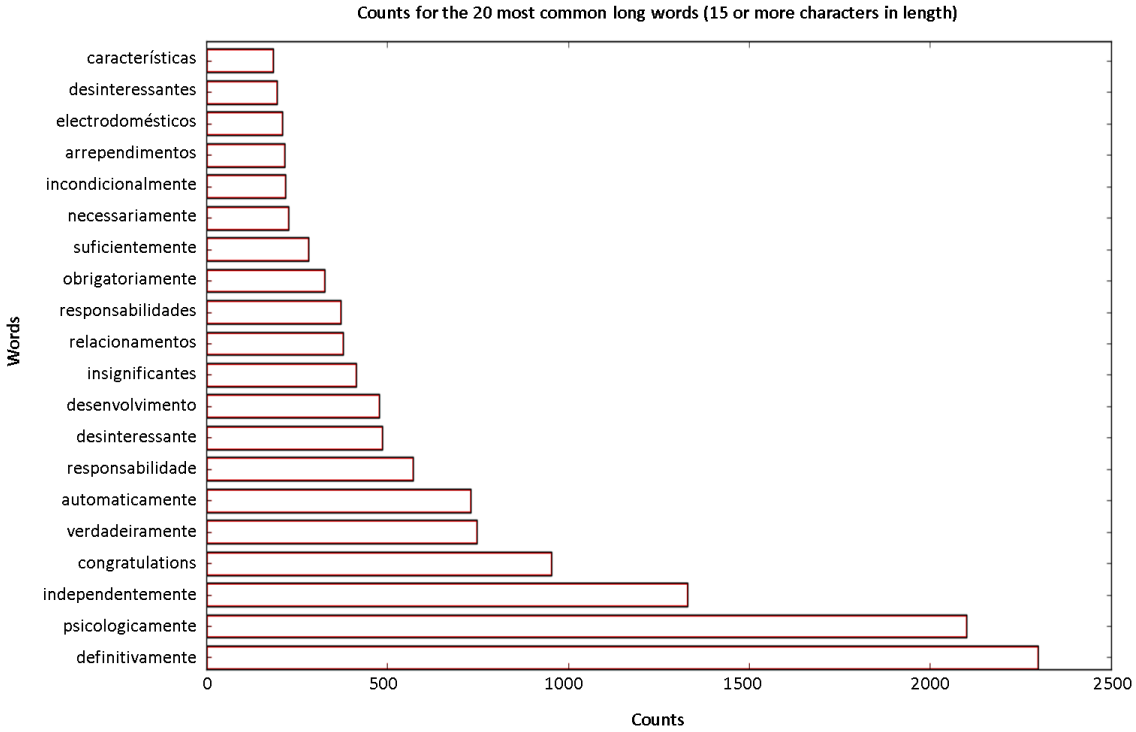

Figure 22: Distribution over the counts of the 20 most common long words found in the tweets collected in 2016.

Table 11: List of the 50 most common long words.

50 most common long words
definitivamente, psicologicamente, independentemente, congratulations, verdadeiramente, automaticamente, responsabilidade, desinteressante, desenvolvimento, insignificantes, relacionamentos, responsabilidades, obrigatoriamente, suficientemente, necessariamente, incondicionalmente, arrependimentos, electrodomésticos, desinteressantes, características, desnecessariamente, pressentimentos, torrencialmente, maioritariamente, homossexualidade, espacoexibicionista, albergaria-a-velha, particularmente, disponibilidade, teleperformance, desesperadamente, montemor-o-novo, profissionalismo, insignificância, especificamente, segundas-feiras, lisboafashionweek, primeiro-ministro, intercontinental, termoacumuladores, psicológicamente, californication, transformavam-no, pulatattoosestudio, estabelecimento, aproximadamente, montemor-o-velho, individualmente, consequentemente, maravilhosamente

This analysis shows two other noteworthy aspects: the first one being, that despite the 140 characters restriction imposed on the tweets length, some users actually use such words and not their abbreviations and the second one being the high correctness of some of these terms which *primeiro-ministro* as opposed to the incorrect and most usually used form *primeiro ministro* is a

clear example of. This could also be an indication of a more formal type of conversation taking place.

With respect to long words in the scope of this work, no specific measure was taken. Also concerning the choice of 15 as the minimum length required to consider a word as being long, no specific reason was taken into account.

The results of the analysis performed on the most common repetitions, see Figure 23 and also Table 12, revealed that most of these are used to express emotion (e.g. *ahahah*, *ohhh*). Some of these emotions are related to real events, such as a goal in a football match (e.g. *golooo*, *golooooo*). It can also be seen that most of these repetitions are just variations of the same base word (e.g. *diaaa* and *diaaaa*).

Similarly to abbreviations, repetitions could simply be removed or in the case of repetitions clearly related to real events, proceed to the normalization of the repetition to its original form (e.g. change *golooo* to its correct form *golo*). Again this would be a difficult task to perform, as some repetitions are legal (e.g. *golooo* would be properly normalized to its correct form *golo*, *Moore* however as in *Roger Moore*, a famous actor who played James Bond, would be incorrectly transformed into *More* thereby completely losing its original intended meaning).

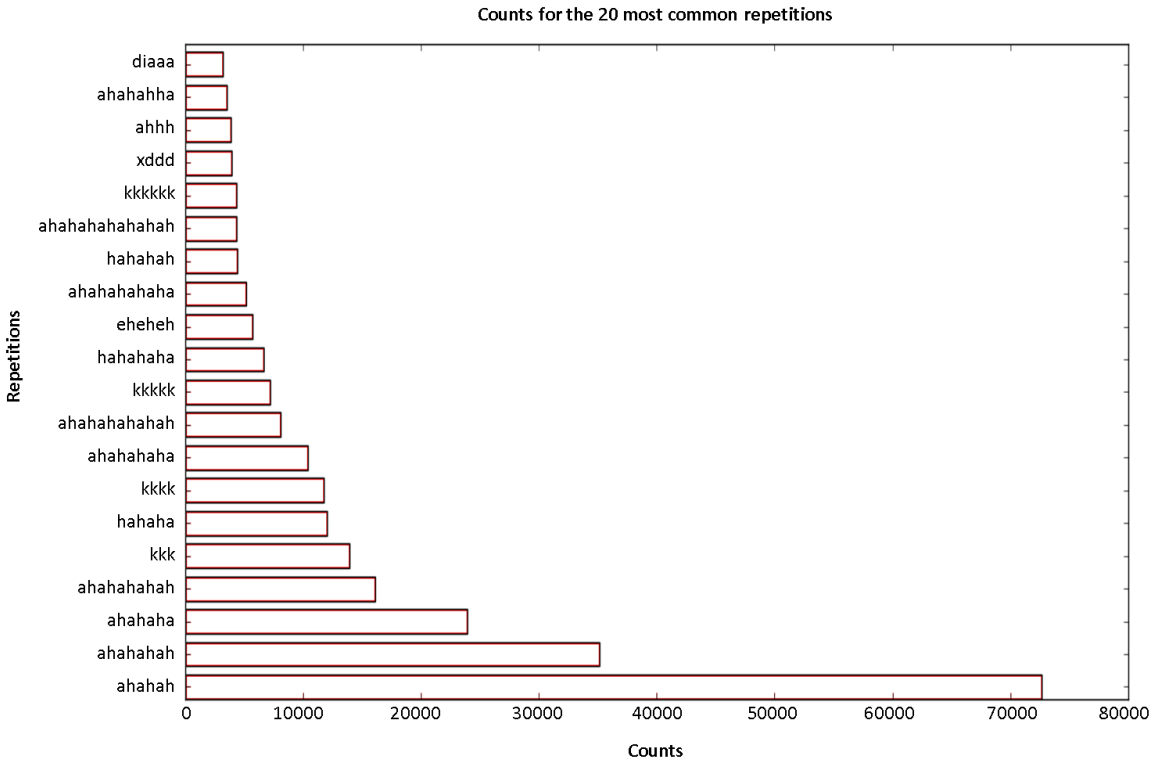

Figure 23: Distribution over the counts of the 20 most common repetitions found in the tweets collected in 2016.

Table 12: List of the 50 most common repetitions.

50 most common repetitions
ahahah, ahahahah, ahahaha, ahahahahah, kkk, hahaha, kkkk, ahahahaha, ahahahahahah, kkkkk, hahahaha, eheheh, ahahahahaha, hahahah, ahahahahahahah, kkkkkk, xddd, ahhh, ahahahha, diaaa, ohhh, kkkkkkk, fdsss, goloooo, hahahahaha, aiii, ahahahahahahahah, ahahahahahaha, golooooo, kkkkkkkk, hahahahah, ahhhh, golooo, golooooooo, fdssss, loool, ahahahahha, xdddd, diaaaa, kkkkkkkkk, kakaka, golooooooo, hehehehe, eheheheh, ohhhh, hmmm, ahahahhaha, ahahahahahahahahah, *, looool

Given this, it was decided that all repetitions comprising one or more characters, occurring in succession more than two times, would be removed. This way correct words and expressions such as *correr* and *ahah* could still be preserved.

For completeness, an analysis over the hapaxes, that is, words that occur only once, was also performed. An assortment of grammatical mistakes, made-up words, words with character repetitions and word combinations, resulting from tokenization issues, can be found in the resulting list. Concerning tokenization issues, it should be noted that the tokenizer used, namely TweetTokenizer, a tokenizer integrated in NLTK, is specific to tweets. An abbreviated list with some of these examples is presented in Table 13. Hapaxes were not removed.

Table 13: List of examples of hapaxes.

Examples	Category
edeficios, tristeteza, teezers	Grammatical mistake
malandrok, maigodr	Made-up words
huehuehhuhue	Character repetitions
numa.boa	Tokenization issues

Table 14: Tweet content summary statistics.

Measure	Description	Value
Links	Average number of links per tweet	0.18
Retweets	Percentage of retweets	0.02%
Replies	Percentage of replies	21%
User Mentions	Average number of user mentions per tweet	0.029
Hashtags	Average number of hashtags per tweet	0.093
Emoji	Average number of emoji per tweet	0.29
All Caps	Percentage of tweets written entirely in all caps	3%
Word Length	Average length of the words used in the tweets	5
Tweet Length	Average number of words composing a tweet	5

To conclude the analysis, several summary statistics concerning tweet content were computed and are presented in Table 14. Amongst the results, some interesting facts are worth

mention, namely the high percentage of replies (21%) and the somewhat unexpected percentage of tweets written entirely in all caps (3%). It can also be seen that on average it is expected that a tweet contains 5 words of length also 5. The Python script used to compute these statistics is presented in Listing 9.

```
PYTHONPATH=~/Desktop/Dissertation python3 computeTweetsStats.py -d 'dumpdb' -c 'dump' -w '2016-01-01T00:00:00' -u '2017-01-01T00:00:00' -v 1
```

Listing 9: Command shell used to compute the statistics related to tweet content, concerning 2016.

5.3 Data Source Infrastructure

Having concluded the exploratory analysis of the dataset, two more decisions remained: 1) what information should be retained from each tweet and in which format; 2) how should this information be persisted. These decisions in turn, dictated the implementation of the Data source Infrastructure, responsible for the pre-processing and storage of the dataset. These implementation choices are discussed next.

5.3.1 Processing

In terms of its structure, each tweet collected is comprised of a set of fields, containing a plethora of different information. These fields in turn are grouped in a hierarchical format that logically separates the information into four main chunks: the information about the tweet itself composes the root level of the hierarchy, while three embedded sections, namely, *place*, *entities* and *user*, group the information about the place, the user and the entities present in the tweet (i.e. links, hashtags and user mentions) respectively. An example of a collected tweet can be found in Appendix E, Listing 19. For reference purposes Table 15 presents some of these fields and their description. The full detailed description of all the fields can be found in [108].

Table 15: Description of some of the fields found in a collected tweet.

Field Name	Description
created_at	UTC time when this tweet was created
Id	The integer representation of the unique identifier for this tweet
retweet_count	Number of times this tweet has been retweeted
Text	The actual UTF-8 text of the status update
user:id	The integer representation of the unique identifier for this user

In order to simplify the structure of the tweet, it was decided to flatten this hierarchy into a single level (an example of a flattened tweet can be found in Appendix E, Listing 20). It was also decided to keep as much of the information contained in the tweet as possible. Some fields however were removed due to redundancy or deprecation, see Table 16. One consequence of this

decision of course, is its higher storage cost. This decision was taken however, in order to keep the information available in case it proved necessary, due to future or unforeseen reasons.

It was also decided that several new fields should be created for each tweet, in order to store some of the summary statistics already presented in Table 14, concerning tweet content (i.e. the text of the tweet). This step was taken to avoid the cost of having to recompute these statistics every time an analysis should be performed later on. These fields are described in Table 17.

Table 16: Fields removed from the tweets.

Field	Removal reason
id_str	Redundant with field id
in_reply_to_status_id_str	Redundant with field in_reply_to_status_id
in_reply_to_user_id_str	Redundant with field in_reply_to_user_id
user_id_str	Redundant with field user_id
geo	Deprecated
contributors	Deprecated
user:following	Deprecated
user:notifications	Deprecated

Table 17: List of the fields added to hold several summary statistics related to the content of each tweet.

Field Name	Description	Type
n_links	The number of links occurring in the tweet	integer
n_users	The number of user mentions occurring in the tweet	integer
n_htags	The number of hash tags occurring in the tweet	integer
n_emojis	The number of emoji occurring in the tweet	integer
all_caps	Whether the tweet is written in all caps	{0,1}[2]
is_reply	Whether the tweet is a reply (i.e. a user mention occurs at the beginning of the tweet)	{0,1}
is_retweet	Whether the tweet is a retweet (i.e. RT occurs at the beginning of the tweet)	{0,1}
has_char_reps	Whether the tweet has words with character repetitions	{0,1}
norm_text	The original content of the tweet (i.e. its text) with links, user mentions, hashtags and emoji removed and with the words normalized to lowercase and their non-accentuated form[3]	text

[2] 0 meaning the false case (i.e. no) and 1 meaning the true case (i.e. yes).

[3] Stopwords and punctuation were not removed at this stage in order to avoid creating false *n*-grams, specifically in the case where $n > 1$, later on.

5.3.2 Storage

The dataset provided consisted of a zipped csv (Comma Separate Values) file with approximately 11.7 GB in size, containing a dump of all the tweets collected until the 30th of June of 2017. This format however was not well suited, as performing any kind of analysis would imply processing the whole file sequentially. This would of course be a very slow, inefficient and computationally expensive process. A better solution would be to process and persist the dataset into a format that could take advantage of operations such as sorting, aggregation, indexing, etc.

As most, if not all database systems, provide some kind of support for the operations just mentioned, using a database to store the dataset was therefore a natural choice. Furthermore, it was decided that a JSON based document database would be most suitable for this purpose.

This decision as to do with the fact that: 1) the tweets collected were originally persisted in JSON format, therefore having native support for this format at the database level would be advantageous (i.e. no need to convert between formats); 2) as the real information needs in terms of the task proposed could not yet be fully assessed at this point and also to preserve as much of the information as possible, for the purposes of future analysis, it was decided to keep most of the information contained in each tweet.

It made sense, therefore, to store the tweets as documents, as this would allow the information related to each tweet to be kept as close as possible, simplifying the access to this same information later on (as opposed for example, of being dispersed into several different tables as is the case with relational databases). Document based databases were introduced in Section 2.5.2.

In this regard, there are several database products of this family to choose from, two of which were already mentioned (i.e. Couchbase and MongoDB), although Couchbase was not discussed in detail. These were then the two candidates put to consideration to be used as the dataset backend.

The immediate reason to choose MongoDB over Couchbase was due to the fact that the system to be implemented was to be hosted in a virtual machine environment running Ubuntu 16.04 LTS, and while MongoDB 3.4 (Community and Enterprise) already included Ubuntu 16.04 in its list of supported platforms [109], Couchbase (Community Edition 4.5.1 and Enterprise Edition 4.6.2) did not (only up to Ubuntu 14.04) [110]. Therefore and to prevent any kind of unforeseen incompatibilities with the operating system, MongoDB 3.4 Community presented itself as the safest option to take.

The minimum system requirements for MongoDB are presented in Table 18. No extra configuration was needed, as the system was to be hosted in a local virtual machine. MongoDB was presented in Section 2.5.2.

Table 18: MongoDB 3.4 Community system requirements.

Component	Minimum Values
Architecture	x86_64/amd64
	s390x[4]
	POWER8 (little endian)[4]
	ARMv8-A
CPU	1[5]
RAM	256 MB[5]
Filesystem	XFS or EXT4 (for Linux)[5]

5.3.3 Database Statistics

The resulting database, named *dumpdb*, after the dataset processing and persistence phases, reached approximately 33.5 GB in size, see Listing 10, with a total count of approximately 29 million documents (i.e. processed tweets), see Listing 11. A more detailed view of the database statistics can be found in Appendix E, Listing 21.

```
show databases
admin      0.0 GB
dumpdb     33,459 GB
local      0.0 GB
```

Listing 10: MongoDB databases listing.

```
db.dump.count()
28,935,607
```

Listing 11: Total number of tweets stored in the database instance.

```
db.dump.createIndex( { created_at: 1, user_id: 1 } )
db.dump.createIndex( { created_at: 1, has_char_reps: 1, n_links: 1, all_caps: 1, n_emojis: 1, n_htags: 1, n_users: 1, is_reply: 1, is_retweet: 1, user_id: 1}, {name: 'features_index'} )
```

Listing 12: MongoDB shell commands used to create the collection indices.

```
PYTHONPATH=~/Desktop/Dissertation python3 parseDatasetDump.py
'/home/developer/Desktop/dump.csv.gz' -d 'dumpdb' -c 'dump' -n 0 -b 100000 -s 0 -v 1
```

Listing 13: Command shell used to process and persist the dataset to MongoDB.

Finally and to improve database query performance, it was decided that two indexes should be created: one index on both the *created_at* and the *user_id* document (i.e. the stored tweet) fields

[4] MongoDB Enterprise only.
[5] Values specific to the WiredTiger storage engine.

and a second index comprising all the fields related to the tweet content summary statistics, see Listing 12. The script used to process and persist the dataset is presented in Listing 13. To optimize the process, the dataset was processed and persisted in batches of 100,000 tweets.

Summary

This chapter presented the analysis conducted on the dataset. Both, measures related to the tweets, concerning their content for example, as well as those related to the users that created these tweets, were taken into account during this analysis. This was done in order to obtain a better understanding of the data and leverage that knowledge to make better decisions concerning the use of these data. One immediate conclusion taken from this analysis was the apparent inconsistency with which tweets were collected. This process is however out of the control of this work. The implementation of the Data source Infrastructure block was then presented, giving special focus to the pre-processing step of the dataset and its appropriate storage. The reasons that dictated these implementation choices were discussed in detail.

6 Testing and Results

This chapter presents and discusses the results obtained during the testing phase of the system as well as other performance measures considered relevant. The chapter starts by presenting the various aspects related to the testing setup of the system, namely in terms of the data used, the precomputed values required, the parameterization of the system and the specifications of the deployment environment. Next the results of the tests conducted are presented and discussed. An analysis regarding the performance aspects of the system is also presented. The overall discussion of the main properties of the system concludes the chapter.

6.1 Data

As already mentioned in chapter 5, the dataset used consists of a set of tweets created in Portugal and collected from the Twitter Search API by a JAVA agent. For the purposes of this work. two subsets of this dataset were used, specifically:

- data collected in the period ranging from the 14th of May of 2015 to the 24th of June of 2015 (3,581,466 tweets) were used in order to tune, train and test the SVM model, as explained in more detail in Section 4.3.4;
- data collected in the period ranging from the 1st of July of 2016 to the 30th of September of 2016 (4,770,636 tweets) were used to test the system.

These periods were chosen due to the fact that data spans several different months, tweets were collected in a consistent way (i.e. no considerable drops occur in the number of collected tweets) and also due to the occurrence of some noteworthy events, such as the 2015 Copa America, the UEFA EURO 2016 and the 2016 Summer Olympics.

The annotation of the data used to tune, train and test the SVM model, alluded to in Subsection 4.3.4 was performed by the author himself. This however, can be considered as being highly prone to bias and in a real production system should obviously be executed by a set of independent annotators. Nevertheless, a method of annotation was followed, in an attempt to

attenuate this bias effect as much as possible, by using a consistent method. The annotation method used consisted of the following guidelines:

- the candidate event should be considered as being related to a real-world newsworthy event when enough evidence can be gathered from its respective segments and otherwise, when such evidence cannot be identified;
- if the candidate event seems clearly related to more than one real-world event, two cases may occur. If one of the events seems to be clearly more represented than the other, then the candidate event should be considered as referring to that event. If no such distinction can be made, then the candidate event should be considered not to be related to any of the events.

Some examples of labeled candidate events, along with the rationale used to obtain the respective labeling decision, are presented in Table 19. As previously stated, the values *1* and *0* were used to label respectively, a candidate event considered to be related to a real-world newsworthy event and otherwise (column *Label*). The _ symbol was used to concatenate the words composing the *n*-grams, with $n > 1$, to help better distinguish the bigrams and trigrams computed during the segmentation phase.

Table 19: Examples of candidate events manually labelled.

Event Segments (candidate event)	Rationale for the labeling	Label
espanha musica suecia alemanha australia russia italia ganhar cantar europa austria albania belgica lugar israel eurovisao	The segments seem to clearly relate this event to the Eurovision musical contest	1
porto bora paulo_lopes andebol malta sporting golo futsal jonas rio_ave campeao campeoes nani epoca hepta marcar ronaldo golos	The segments seem to allude to several different events. This makes it hard to discern to which real event to actually relate this candidate event to	0
futsal jogo sporting golo	The segments seem to clearly relate this event to a futsal game	1
american_pie argentina ver filme jogo tvi paraguai copa_america filmes filme_de_terror ver_o_jogo ver_um_filme mamma_mia	Some segments seem related to a football game between Argentina and Paraguay in Copa America. However, most of the other segments are clearly unrelated to it	0
irma prima primo namorado tia avo	The segments are clearly not related to any event	0

6.2 Precomputed values

Two of the precomputed values required by the system, namely the segment probabilities and the segment frequency probabilities, were computed and setup, as discussed in Section 4.4.2 and Section 4.4.4 respectively, using the data from the testing period just mentioned (i.e. 1st of July of 2016 to the 30th of September of 2016). The TF-IDF model mentioned in Section 4.3.3 and used to

compute the similarity of the pseudo-documents associated to the segments, was also fitted (i.e. learned the vocabulary) using these same data. The Wikipedia anchor probabilities were computed and set up using a Wikipedia dump as detailed in Section 4.4.3.

6.3 Parameterization

In terms of the parameterization of the system, the values chosen mimic those proposed in the original system, specifically, the size of each time window *t* was fixed to be a whole day, being further sub-divided into sub-time windows *t'* of 2 hours, and the values used for *K* and *k* were $\sqrt{N_t}$ and 3 respectively. These values are listed in Table 20.

Concerning the value chosen for the time window period S_t (i.e. one day), in terms of event detection, this choice implies that we are interested in identifying events that occur within a day. As this value can be parameterized, other values could have also been used (e.g. half a day, two days, etc), in order to better reflect the intended purpose in terms of the duration of the events of interest to be identified by the system.

Table 20: Parameterization used to test the system.

Parameter	Description	Value
S_t	The size of time window *t*	1 day
K	The top-k bursty segments to retain	$\sqrt{N_t}$
k	The k-nearest neighbors to consider for Jarvis-Patrick clustering	3
S_m	The size of sub-time window *t'*	2 hours

6.4 System Deployment

The implemented system was deployed in a virtualized guest environment, running Ubuntu 16.04 LTS with 2 allocated processor cores, 5 GB of RAM and 80 GB of disk space. VMware Player was used as the virtualization agent. In order to avoid potential problems associated with server downtime and network related access permission issues, the whole system was deployed in this environment.

6.5 Results

The results were obtained via the following procedure: first the system was used in order to compute the events for each testing period considered, retaining both the candidate events before and after the filtering step. These were then manually inspected as follows: the candidate events obtained prior to filtering, were manually labeled as being related to real-world newsworthy events or not. This was done in order to obtain *Me*, the total number of unfiltered candidate events found

by the system and related to real-world newsworthy events. *Me* in this case can be interpreted as an approximation to the real number of real-world events present in the data.

The candidate events obtained after the filtering step (i.e. the final events computed by the system), were in turn inspected in order to calculate the number of correct *Te* (i.e. candidate events classified correctly as real events or the true positives) and incorrect *Fe* (i.e. candidate events incorrectly classified as real events or the false positives) classifications respectively. These values were then used to derive the *precision* and *recall* measures of the system. *Precision* and *recall* were computed respectively as $precision = \frac{Te}{Te+Fe}$ and $recall = \frac{Te}{Me}$.

It should be noted that candidate events considered to be related to the same real-world event were counted independently, in order to simplify the process (i.e. two candidate events related to the same real-world event count as two correct classifications as opposed to just one). It should also be noted that the values obtained by these measures (i.e. *precision* and *recall*) are just an approximation of the real values, as manually inspecting all tweets in the dataset, in order to derive the real number of newsworthy events present in the data, would be unfeasible. Table 21 lists these results for each period tested, where each column represents the following:

- column T. CE (total candidate events) denotes the number of candidate events computed prior to the filtering step;
- column M. CE (Manual candidate events) denotes the number of candidate events obtained prior to the filtering step, found to be related to real-world newsworthy events, after manual inspection;
- column F. CE (filtered candidate events) denotes the same as the M. CE column, with the exception that the candidate events inspected were the ones obtained after the filtering step (the number of correct classifications and misclassifications respectively, is also shown in parenthesis);
- column Prec, denoting precision and column recall, are related to those same measures of performance of the system;
- column N. Tweets (Number of tweets) shows both the total number of tweets processed as well as the average number of tweets processed per time window *t* for each of the periods tested.

The list comprising all the events obtained for the period between 2016-07-01 and 2016-07-31 is also depicted in Table 22. Again, the _ symbol was used to concatenate the words composing *n*-grams with $n > 1$. The order in which the segments are shown is directly associated to their weight rank (segments with higher weight are listed first in order). Swear words were also elided. The script used to run the system and obtain the final events is presented in Listing 14.

Table 21: Test results.

Period	T. CE	M. CE	F. CE	Prec.	Recall	N. Tweets
2016-07-01 – 2016-07-31	894	57	31(27/4)	87%	47.4%	1,650,097 / 53,229
2016-08-01 – 2016-08-31	961	50	22(16/6)	72.7%	32%	1,673,762 / 53,992
2016-09-01 – 2016-09-30	786	37	25(17/8)	68%	45.9%	1,446,777 / 48,226
Total	**2,641**	**144**	**78(60/18)**	**76.9%**	**41.6%**	**4,770,636**

Table 22: List of events obtained for the period between 2016-07-01 and 2016-07-31.

Id	Segments of the event	Description
e1	pais_de_gales islandia jogar gales contra vs ramsey	Related to UEFA EURO 2016 Wales national football team
e2	alemanha italia islandia bora final buffon ganhar muller ganha	Related to UEFA EURO 2016, Germany vs Italy match
e3	penalti penaltis boateng falha marcar ozil	Related to e2, penalties decision
e4	islandia franca alemanha final euro contra	Related to UEFA EURO 2016, discussing three of the possible final candidates
e5	pais_de_gales gales barcelos equipa polonia	Related to UEFA EURO 2016, Portugal vs Wales semi-final match
e6	renato_sanches quaresma andre_gomes joao_mario renato joao_moutinho entrar rabo campo marca adrien entra	Related e5
e7	alemanha franca vamos ganhar venha	Related to UEFA EURO 2016, discussing adversaries of Portugal in the final
e8	torre_eiffel portugues orgulho portuguesa enorme emocao bandeira	Related to e5 and Portugal's victory in the match
e9	final golo tamos_na_final rumo_a_final gritar tamos	Related to e5 and e8
e10	portugal alemanha franca final ganhar contra ganhe vs	Related to UEFA EURO 2016, discussing possible adversaries of Portugal in the final and the Germany vs France semi-final match
e11	radiohead tame_impala foals alive alges curtir rtp vivo	Related to Radiohead performing at NOS Alive Festival
e12	pixies robert_plant wolf_alice biffy_clyro	Bands playing at NOS Alive Festival
e13	radiohead tame_impala hot_chip karma_police foals creep amazing last_night	Related to e11
e14	arcade_fire alive alges band_of_horses	Bands playing at NOS Alive Festival
e15	tou aqui bue farta curti	Miscellaneous of common use words
e16	vou amanha dormir ver_o_jogo acordar terreiro	Miscellaneous of common use words

e17	rui_patricio lisboa patricio terreiro_do_paco marques_de_pombal eder campeoes fernando_santos golo deus europa coracao aguenta_coracao festa ganhamos terreiro minutos bola festejar rua campeoes_da_europa campeoes_europeus penaltis marcar alameda homem_do_jogo voz entrar gritar duvida marca marques aguentar prolongamento poste baliza	Related to UEFA EURO 2016, Portugal vs France final match
e18	bruno_alves payet	Related to e17. Several insultuous words were removed. This was due to the violent foul committed by Payet, injuring Cristiano Ronaldo
e19	franca franceses jogo jogar neste sujo	Related to e17 and e18
e20	cristiano_ronaldo ronaldo frances capitao treinador jogador melhor_do_mundo cristiano joelho saiu	Related to e17, e18 and e19, alluding to the injury of Ronaldo
e21	bora la vamos ganhar le bora_bora c'est au je_suis	Although related to e17 was not considered due to the fact that several other segments present are meaningless
e22	hoje europa ontem campeoes eder jogo disse ia campeoes_europeus feriado golo bom_dia campeoes_da_europa	Related to the victory achieved by the Portuguese national football team in the UEFA EURO 2016
e23	lisboa marques_de_pombal terreiro_do_paco avenida_dos_aliados aeroporto_de_lisboa selecao aeroporto humberto_delgado eusebio borboleta chegar incrivel marques aviao montijo terreiro aliados ambiente	Related to e22. Arrival of the Portuguese national football team
e24	renato_sanches cristiano_ronaldo ronaldo chorar nani renato treinador bola_de_ouro beatbox	Related to e23
e25	iron_maiden meo_arena	Related to the announced performance of Iron Maiden at MEO Arena
e26	figueira_da_foz hardwell voltar dj_snake sunset borgore rfm_somnii rfm figueira somnii kshmr	Related to several bands performing at the RFM Somnii 2016 festival
e27	eder alta_definicao entrevista_do_eder entrevista	Related to e22. Éder, the player who scored the winning goal in the EURO final, gives an interview at TV show Alta Definição
e28	hoquei_em_patins hoquei campeoes_europeus campeoes campeoes_da_europa	The Portuguese national roller hockey team wins the 52nd edition of the European Roller Hockey Championship (CERH)
e29	andre_gomes barcelona barca milhoes	Football player André Gomes transfers to Barcelona
e30	ir vou hoje dormir sair	Miscellaneous of common use words

e31	mundo acabar acaba fim_do_mundo acabe supostamente	Talking about the end of the world date announced to happen in the following day by some prophecy

```
PYTHONPATH=~/Desktop/Dissertation python3 tweventBasedEventsDetector.py -d 'dumpdb' -c 'dump' -w '2016-07-01T00:00:00' -u '2016-08-01T00:00:00' -i 'trained_models/tfidf_vectorizer_07_09_2016' -s 'trained_models/event_classifier' -l 1 -v 1
```

Listing 14: Command shell used to run the event detection process.

6.6 Discussion

This section discusses the results obtained during the testing phase of the system. The overall results obtained are first presented. Then the several periods tested are compared in terms of these results. A comparison with the results obtained by similar implementations is also presented. Finally, other relevant aspects, such as those related to the performance of the system and to the quality of the textual representation of the events obtained, are also addressed.

6.6.1 Overall Results

In terms of the overall results obtained, as depicted in Table 21, the system presents a somewhat reasonable *precision* of 76.9% but a fairly low *recall* of 41.6%. It can also be seen that these values vary considerably amongst the different periods tested, ranging from 68% to 87% in the case of *precision* and from 32% to 47.4% in the case of *recall*. Some variation can also be observed regarding the number of real events manually identified prior to the filtering step (the values shown in the M. CE column), with a clear drop during the third period tested, corresponding to September with 37 real events identified.

Regarding the overall results obtained in terms of *precision* and *recall*, several explanatory reasons can be enumerated. One reason for this, may be the fact that the features selected to train the SVM model, may not be sufficient or representative enough in order to distinguish appropriately the candidate events related to real-world newsworthy events, from those not related to these events, therefore heavily penalizing *recall* and also *precision* to a lesser extent, in which case other or more representative features should be further derived.

Another possible reason may stem from the fact that the training dataset used was imbalanced, hindering the learning of the model. In fact there seems to be some over-fitting effect as the testing accuracy of 92% obtained during the training phase is much higher than that obtained with new data, 76.9% in this case.

Further undersampling the training dataset (i.e. remove candidate events considered as not related to real-world newsworthy events), in order to obtain a similar number of examples for both cases, or obtaining more training data, could potentially help alleviate this problem. Of course that

undersampling has its own issues, as relevant information may be lost during the process of removal of these samples.

Yet another reason may be related to incorrect manual labeling of the training dataset, as this can be a source of disturbance during the training phase of the model, by introducing unwanted noise in the learning process, which in turn may cause the model to learn poorly from this lower quality training data, affecting its classification performance and generalization power (i.e. the ability to properly classify yet unseen data).

Using several different annotators to annotate the training data and deciding the final label of each of the candidate events according to a strategy of majority vote, may help alleviate this issue. Of course that, if the volume of candidate events to be annotated or labeled manually is considerable, as in this case, it may prove difficult to implement such a strategy, due for example to time or man power constraints.

Finally, the somewhat low quantity of tweets collected for the periods tested, may also affect the performance of the system, as many tweets related to events may not have been collected.

6.6.2 Inter-Period Comparison Results

Concerning the differences observed in the *precision* and *recall* results obtained for the different periods tested, several possible reasons can also be enumerated. One such reason may be due to insufficient training data, as not all types of events can be covered and these in turn may be characterized differently in terms of the features selected to train the model.

Regarding this point, it should be noted that in terms of real-world events, the period of data used as the training data for the SVM filtering model, namely from the 14th of May of 2015 to the 30th of June of the same year, was mostly dominated by end of season football matches, the UEFA Champions League and the Copa America.

In a similar regard, the data used to test the system can be characterized as follows: July was mostly dominated by the UEFA EURO 2016 as well as several summer festivals. August was dominated by the 2016 Summer Olympics and several major fires that occurred all over the country. September seemed mostly dominated by football matches both national as well as international and the return to school.

It is therefore possible that the characteristics of the different types of real-world events are also reflected differently, in terms of the values of the features of the respective candidate events obtained, according to their impact or nature. As an example of this the UEFA Champions League event may be more related to the UEFA EURO event due to their similar nature then to the Summer Olympics. This in turn could explain the reason why July obtained the best performance measures for both *precision* and *recall*. Choosing time periods where the most variety of these

events occurred, as the training data of the model, may attenuate this issue. That may however not be possible due to insufficient data, as in the case of the dataset used in this work.

In the case of the results obtained for the period concerning September, the clear drop in terms of *precision*, when compared with the other periods, may be related to the fact that as already alluded to, no major events occurred during that period.

6.6.3 Comparison with the Reference Systems

Table 23 depicts how the results obtained by the system implemented in this work compare to the results obtained by other implementations of the same base system, namely: Twevent, the system used as the base of this implementation and FRED.

Overall, the system implemented in this work detected much less real event, only 78, when compared to the other two systems, achieved a lower *precision* and a higher *recall* (Twevent is excluded from the analysis on *recall*). A few remarks should be noted however.

With respect to Twevent, although the overall number of tweets used to test the system is roughly the same, it should be noted that the data used in Twevent concerns a single month, while the data used to test this system, concerns three times as much, that is, three months. Also, the dataset used in Twevent was built purposely to test that system, using a specific procedure, fully detailed in [28]. Some of the most noteworthy events occurring in this dataset are the FIFA World Cup 2010, the WWDC 2010, and the MTV Movie Awards 2010.

While the data used to test the system implemented in this work also contains examples of such events, such as the UEFA EURO 2016 and the 2016 Summer Olympics, no specific procedure was used to collect the tweets.

Finally, Twevent did not use a model to perform the filtering process and therefore reports the *recall* as the number of distinct realistic events detected, which in turn means that no comparison can be made regarding this measure of performance. This value is therefore not presented in Table 23.

With regard to FRED, the data used to test that system is almost eight times the data used to test the system presented in this work and covers only 15 days. Regardless, FRED obtained a higher *precision* but a lower *recall*.

Regarding the importance of the features used to train the SVM model, since this work uses a subset of the features proposed in FRED, it would be interesting to compare the findings of the two works. Similarly to FRED, the features found to be more relevant in training the SVM model to perform the filtering step were *wiki*, *sim* and *tag*. FRED also reports *url*, which in the case of this system is ranked below *mem*, but this is not really relevant as only the first three features

mentioned, truly differentiate themselves from the rest The analysis performed on the importance of these features was presented in Section 4.3.4.

Table 23: Comparison results.

System	#Events	Precision	Recall	Data period	N. Tweets
Twevent	101	86.1%	--	June 2010	4,331,937
FRED	146	83.64%	22.89%	1-15 January, 2013	31,097,528
This system	**78**	**76.9%**	**41.6%**	1st July–30th September, 2016	**4,770,636**

6.6.4 Quality of the Events Obtained

With respect to the quality of the textual representation of the real events obtained, as well as their purity, that is, the amount of event segments present in the textual representation of the event effectively related to the identified real event, the following can be observed from Table 22:

- the occurrence of several bigrams and also trigrams denoting several types of entities such as the name of people (e.g. *andre_gomes* in *e29*, *cristiano_ronaldo* in *e24*), places (e.g. *figueira_da_foz* in *e26*) or musical bands and artists (e.g. *iron_maiden* in *e25*, *dj_snake* in *e26*) for example, provides a more informative and descriptive representation of the event as compared to single words. This also allows the events to be more easily interpreted, as the context as well as some of the actors involved can be identified more easily.
- while most of the real events obtained and listed can be considered fairly pure, as most of the event segments that textually represent them, are related to that same event (e.g. *e28*, *hoquei_em_patins hoquei campeoes_europeus campeoes campeoes_da_europa*), some of them present a few unrelated segments in their textual representation (e.g. *e22*, *hoje europa ontem campeoes eder jogo disse ia campeoes_europeus feriado golo bom_dia campeoes_da_europa*, where the event segments *hoje*, *disse*, *ia*, *feriado* and *bom_dia* seem clearly to be unrelated to the identified event). This impurity also contributes to adding noise to the features computed from these candidate events and may cause the manual labeling to be difficult to judge in some cases, due to the mixed context present in the textual representation of the candidate events.

6.6.5 System Performance

In terms of the performance of the system regarding processing time, it can be seen that the components presenting the biggest bottleneck, are the Event Segment Clustering component and the Tweet Segmentation component, taking on average 1.28 and 1.03 minutes to compute, respectively. The remaining two components represent a residual factor in this regard. On average a

time window (approximately 51,816 tweets per time window) took 2.32 minutes to compute. These values are depicted in Table 24.

Given these results, the Event Segment Clustering and the Tweet Segmentation components, should therefore constitute the first optimization effort in order to improve the scalability of the system. In this regard, the tweet segmentation task, performed by the Tweet Segmentation component, could be parallelized, as the segmentation of the tweets can be performed independently.

It should be noted that the design of the system allows both parallelization as well as distributed techniques to be applied in order to improve its performance. This pipeline in particular, was setup to be able to work using threading. However the fact that the virtual machine where the system was deployed had only 2 cores available and also due to internal specificities of the implementation of Python, discussed in more detail in [111], that do not allow for the true parallelization of the threads, this ended up not being a factor of any significant improvement in the performance of the system.

Table 24: Components average running times.

Total time	Tweet Segmentation	Event Segment Detection	Event Segment Clustering	Event Filtering
2.32 m	1:03 m	0.59 ms	1.28 m	0.033 ms

6.7 Overall System discussion

Regarding the results just presented and discussed, it may therefore be also of interest to address the applicability of this system to the task proposed, as well as some of the advantages and disadvantages inherit to its design properties.

In terms of deployment, the system presents some complexities, as at least three precomputed values are required and must be therefore internally managed or obtained via the integration with external services. Regarding the computation of these precomputed values, the following additional aspects should be noted:

- calculating and storing these values can be costly in terms of both time and storage space required. As an example, processing Wikipedia took almost 9 hours and calculating the segment probabilities around 1.30 hours. The Redis instance that stored these values occupied almost 1 GB in memory. This cost increases obviously, as more data is used to perform these calculations.
- due to its internal implementation, the system is very susceptible to these values and the results obtained may vary considerably, depending on the method used for their calculation.

The parameterization of the system may also prove challenging, as four parameters must be setup and their values may also impact the performance of the system. On the other hand, the design of the system and the simplicity of the processes used to perform the detection of the events, make it very scalable, as parallelization and distributed techniques can easily be used.

To conclude, the system can be used to perform the task proposed, however its performance may be to susceptible to the factors just mentioned, meaning that a greater effort must be spent dealing with the details associated to its parameterization and setup required in terms of the pre-calculated values mentioned.

Summary

This chapter presented and discussed the results obtained during the testing phase of the system. These results were then compared with those obtained by similar implementations. Other relevant aspects of the system, such as its performance and the quality of the textual representation of the events obtained, were also topics of this discussion. Overall, the system implemented achieved a lower *precision* but a higher *recall*, when compared to similar implementations. Some significant differences between the several periods tested, in terms of the results obtained, were also identified. The possible reasons that explain these results were presented. Amongst these, the lack of data and the possible overfitting effect alluded to in the discussion, are two main highlights.

7 Conclusion

This chapter presents an overall overview of the work accomplished in this dissertation. The chapter starts by presenting the main challenges and difficulties encountered during the implementation of the event detection system proposed. Next, some improvements, changes or additions are proposed in order to further enhance the system implemented, in terms of both its accuracy and operation performance. The discussion of the work accomplished concludes the chapter.

7.1 Main Difficulties and Challenges

The main difficulties encountered, had to do with constraints inherit to the resources of the deployment system, specifically in terms of its available memory. This posed some challenges when computing the precomputed values required, as these computations could not be performed entirely in memory. In the case of Wikipedia, for example, the dump file had to be processed using a MapReduce approach based on intermediate results stored in separate files in the hard drive.

Redis also had to be leveraged, both as the storage backend of these values as well as to support their calculation, as using Python dictionaries to achieve these goals, proved unfeasible. The phased calculation required to compute the segment frequency probabilities is an example of this. The considerable number of these values, as well as the challenges associated to their computation, also hindered the implementation process, as these values had to be in place before the system could be properly tested. Furthermore, the results obtained may vary considerably depending on these values, namely on how they were computed and on which data was used for that computation. This also proved to be an important and difficult aspect of the implementation.

On one hand, the constraints posed by the deployment system made it unfeasible to use the whole dataset in the calculation of for example, both the segment frequency probabilities and the segment probabilities. On the other hand, the tweets were not collected in a consistent way and several periods of data are missing. This made it more difficult to choose the appropriate data to be used.

Regarding the main challenges posed by the implementation in terms of development, it is noteworthy referring the following:

- the tweet segmentation algorithm, implemented using a dynamic programming approach, in order to allow this step to be performed as efficiently as possible;
- the implementation of a variant of the Jarvis-Patrick clustering algorithm, not yet available in Scikit-Learn;
- the tuning, training and testing phases of the SVM model used in the filtering step of the detection process. In this regard, the imbalanced nature of the training data, posed an additional challenge, specifically in terms of trying to avoid the possible overfitting effect introduced by this kind of imbalanced training data.
- the overall tuning of the system in terms of its computational time. Concerning this aspect, an additional effort was required in order to make it possible to lower the computational time of the system, per time window, from almost 20 minutes to only 2.32 (on average). The tuning of the Event Segment Clustering component, played an important role in this regard.

7.2 Future Work

Given the results obtained, a few aspects were identified as possible improvements to the implemented system, in terms of future work. These are listed below:

- the textual representation and purity of the real events obtained as the final result of the system, can be further improved. In this regard, both the Tweet Segmentation component as well as the Event Segment Detection component are of particular relevance. The first of these, because it dictates the quality of the segments obtained which in turn also plays an important role during the similarity testing of the segments and ultimately in the results. As an example of this, the context associated to the segment *Cristiano Ronaldo*, that is, the tweets containing this segment, is expected to be very different from the context associated to the segment *Cristiano* or the segment *Ronaldo*, when taken separately. The context of the segments in turn is an important piece taken into account to determine their similarity and hence group them as being related to the same event. Also, the segment *Cristiano Ronaldo* is much more descriptive. The second of these, because it discards segments, which again has implications in the similarity tests and also in the results. As an example of this, it can be easily seen that segments are tested for similarity not against all the detected segments, but only against a subset of these, the top K in the ranking. This means that the segments determined to be more similar to a given segment may not in fact be those more similar to it in the overall set of segments, as some of these may have been

discarded from further processing, but those amongst the top *K* subset. This may lower considerably the quality of the real events obtained and hinder results.

- the detection of more than one event related to the same real-world event, was not addressed in this work and could therefore be further explored, as this may be a relevant issue in some use cases (e.g. *e11* and *e13* in Table 22 are related to the same real-world event and were detected in different days);
- in this work, parallelization and distributed computing techniques were not explored and could therefore be suitable solutions in order to improve the scalability and performance of the system, in terms of its computational time cost;
- the components detected as more costly, for example in terms of operation time, (i.e. the Tweet Segmentation component and the Event Segment Clustering component) can also be further optimized;
- other machine learning alternatives to SVM can also be tested in order to assess their applicability in performing the filtering step, such as other algorithms or even ensembles (e.g. Random Forest);
- other features related to the candidate events or possibly more descriptive can also be identified, in order to better train these models and improve their performance measures;
- test the implemented system with a different dataset, preferably one containing more data and collected in a more consistent way;
- a more thorough study should be conducted in order to assess the real impact of the change proposed and introduced in the weight scheme used to rank the tweet segments (and obtain the event segments), in terms of the results obtained;
- the computation process used to calculate some of the precomputed values, namely the Wikipedia anchor probabilities and the segment probabilities as well as the options made regarding this computation, should be assessed in order to understand their real impact in the results obtained, as these in turn depend on these values;
- the system should also be tested under several different parameterization settings, in order to understand the importance and the impact of each of these parameters regarding the results obtained.

7.3 Final Considerations

The purpose of this work was to implement an event detection system based on tweets, intended to detect newsworthy real-world events. A similar system already proposed in the literature, chosen for the reasons mentioned during the discussion of the goals in Section 1.2, was used as the base of the implementation. In order to achieve these goals, the system was then fully implemented. Some

of the highlights of this implementation include: 1) the tweet segmentation algorithm implemented, using a dynamic approach; 2) the proposal to use Wikipedia as an additional factor in order to derive a better weighting scheme; 3) the implementation of a variant of the Jarvis-Patrick clustering algorithm; 4) the tuning, training and testing of the SVM model using manually annotated data.

The precomputed values required by this system implementation were also appropriately computed and stored. All the details as well as the rationale behind the choices made during this implementation phase were presented and appropriately discussed.

The system was then tested using three months of tweets created in Portugal and written mostly in the portuguse language, comprising a total of 4,770,636 tweets**.** The results obtained were then presented, discussed and compared to those also obtained by similar implementations. The performance aspects of the system were also addressed during this discussion. The applicability of the system implemented, regarding the task proposed, as well as the discussion of some of its advantages and disadvantages were also a topic of discussion.

To conclude, it can therefore be considered that all the objectives proposed initially were successfully achieved, although the system obtained can still be further improved in several aspects.

References

[1] P. Nicolaos, K. Ioannis, and G. Dimitrios, "Detecting Events in Online Social Networks: Definitions, Trends and Challenges," in *Solving Large Scale Learning Tasks. Challenges and Algorithms. Lecture Notes in Computer Science*, vol. 9580, Springer, Cham, 2016, pp. 42–84.

[2] "Welcome to Twitter." [Online]. Available: https://twitter.com/?lang=en. [Accessed: 30-Sep-2017].

[3] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a Social Network or a News Media?," in *WWW '10 Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 591–600.

[4] A. Java, X. Song, T. Finin, and B. Tseng, "Why We Twitter: Understanding Microblogging Usage and Communities," in *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, 2007, pp. 56–65.

[5] F. Atefeh and W. Khreich, "A Survey of Techniques for Event Detection in Twitter," *Comput. Intell.*, vol. 31, no. 1, pp. 132–164, 2015.

[6] A. Madani, O. Boussaid, and D. E. Zegour, "What ' s Happening : A Survey of Tweets Event Detection," in *INNOV 2014 : The Third International Conference on Communications, Computation, Networks and Technologies*, 2014, pp. 16–22.

[7] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors," in *Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 851–860.

[8] S. Van Canneyt, M. Feys, S. Schockaert, T. Demeester, C. Develder, and B. Dhoedt, "Detecting Newsworthy Topics in Twitter," *CEUR Work. Proceedings, Proc. SNOW 2014 Data Chall.*, vol. 1150, pp. 25–32, 2014.

[9] S. Papadopoulos, D. Corney, and L. M. Aiello, *SNOW 2014 Data Challenge: Assessing the Performance of News Topic Detection Methods in Social Media*. 2014.

[10] C. Li, A. Sun, and A. Datta, "Twevent: Segment-based Event Detection from Tweets," in

Proceedings of the 21st ACM International Conference on Information and Knowledge Management - CIKM '12, 2012, pp. 155–164.

[11] Y. Qin, Y. Zhang, M. Zhang, and D. Zheng, "Feature-Rich Segment-Based News Event Detection on Twitter," in *Sixth International Joint Conference on Natural Language Processing*, 2013, pp. 302–310.

[12] "IBM Watson." [Online]. Available: https://www.ibm.com/watson/. [Accessed: 27-Oct-2017].

[13] "Build on the AI platform for business." [Online]. Available: https://www.ibm.com/watson/developer/. [Accessed: 27-Oct-2017].

[14] F. F. Duarte, D. D. Regateiro, Ó. M. Pereira, and R. L. Aguiar, "On the Prospect of using Cognitive Systems to Enforce Data Access Control," in *IoTBDS 2017*, 2017, pp. 412–418.

[15] "IoTBDS 2018." [Online]. Available: http://www.iotbd.org/. [Accessed: 06-Nov-2017].

[16] A. M. Kaplan and M. Haenlein, "Users of the world, unite! The challenges and opportunities of Social Media," *Bus. Horiz.*, vol. 53, no. 1, pp. 59–68, 2010.

[17] "Number of social media users worldwide from 2010 to 2021 (in billions)." [Online]. Available: https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/. [Accessed: 21-Oct-2017].

[18] "Number of monthly active Twitter users worldwide from 1st quarter 2010 to 2nd quarter 2017 (in millions)." [Online]. Available: https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/. [Accessed: 02-Oct-2017].

[19] "Log In to Facebook." [Online]. Available: https://www.facebook.com/. [Accessed: 03-Oct-2017].

[20] "Number of monthly active Facebook users worldwide as of 2nd quarter 2017 (in millions)." [Online]. Available: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/. [Accessed: 03-Oct-2017].

[21] "Instagram." [Online]. Available: https://www.instagram.com/. [Accessed: 04-Oct-2017].

[22] "Number of monthly active Instagram users from January 2013 to September 2017 (in millions)." [Online]. Available: https://www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/. [Accessed: 04-Oct-2017].

[23] A. Bifet and E. Frank, "Sentiment Knowledge Discovery in Twitter Streaming Data," in *DS'10 Proceedings of the 13th International Conference on Discovery Science*, 2010, pp. 1–15.

[24] M. Mathioudakis and N. Koudas, "TwitterMonitor: Trend Detection over the Twitter Stream," in *2010 ACM SIGMOD International Conference on Management of Data*, 2010,

pp. 1155–1157.

[25] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, “Measuring User Influence in Twitter: The Million Follower Fallacy,” in *Fourth International Conference on Weblogs and Social Media, ICWSM 2010*, 2010, pp. 10–17.

[26] W. Galuba, K. Aberer, D. Chakraborty, Z. Despotovic, and W. Kellerer, “Outtweeting the Twitterers - Predicting Information Cascades in Microblogs,” in *Proceedings of the 3rd Workshop on Online Social Networks (WOSN 2010)*, 2010.

[27] A. Signorini, A. M. Segre, and P. M. Polgreen, “The Use of Twitter to Track Levels of Disease Activity and Public Concern in the U.S. during the Influenza A H1N1 Pandemic,” *PLoS One*, vol. 6, no. 5, 2011.

[28] J. Weng and B. Lee, “Event Detection in Twitter,” in *Fifth International AAAI Conference on Weblogs and Social Media (ICWSM-11)*, 2011, pp. 401–408.

[29] C. C. Aggarwal and K. Subbian, “Event Detection in Social Streams,” in *Proceedings of the 2012 SIAM International Conference on Data Mining*, 2012, pp. 624–635.

[30] A. J. McMinn, Y. Moshfeghi, and J. M. Jose, “Building a Large-scale Corpus for Evaluating Event Detection on Twitter,” in *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management - CIKM ’13*, 2013, pp. 409–418.

[31] H. Becker, M. Naaman, and L. Gravano, “Beyond trending topics: Real-world event identification on Twitter,” in *Proceedings of the Fifth International Conference on Weblogs and Social Media*, 2011.

[32] C. C. Aggarwal and C. Zhai, “A Survey of Text Clustering Algorithms,” in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Boston: Springer, 2012, pp. 77–128.

[33] S. T and V. S. Kumar, “Application of Big Data in Data Mining,” *Int. J. Emerg. Technol. Adv. Eng.*, vol. 3, no. 7, pp. 390–393, 2013.

[34] M. A. U. D. Khan, M. F. Uddin, and N. Gupta, “Seven V’s of Big Data understanding Big Data to extract value,” in *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education, ASEE Zone 1*, 2014.

[35] “Introducing JSON.” [Online]. Available: http://www.json.org/. [Accessed: 29-Aug-2017].

[36] W3C, “Extensible Markup Language (XML).” [Online]. Available: https://www.w3.org/XML/. [Accessed: 29-Aug-2017].

[37] Oracle, “Oracle: Big data for the enterprise,” *Oracle White Paper*. 2013.

[38] K. Lee, B. D. Eoff, and J. Caverlee, “Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter,” in *Proceedings of the Fifth International Conference on Weblogs and Social Media*, 2011.

[39] “Twitter Usage Statistics.” [Online]. Available: http://www.internetlivestats.com/twitter-

statistics/. [Accessed: 30-Nov-2017].

[40] S. Raschka, *Python Machine Learning*, First Edit. Birmingham, UK: Packt Publishing Ltd, 2015.

[41] E. Alpaydin, *Introduction to Machine Learning*, Second Edi. The MIT Press, 2010.

[42] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*. The MIT Press, 2006.

[43] P. Harrington, *Machine Learning in Action*. Manning Publications Co., 2012.

[44] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A Practical Guide to Support Vector Classification," *Technical Report Department of Computer Science, National Taiwan University*. 2003.

[45] S. Developers, "Support Vector Machines." [Online]. Available: http://scikit-learn.org/stable/modules/svm.html. [Accessed: 08-Oct-2017].

[46] R. A. Jarvis and E. A. Patrick, "Clustering Using a Similarity Measure Based on Shared Near Neighbors," *IEEE Trans. Comput.*, vol. 22, no. 11, pp. 1025–1034, 1973.

[47] S. Robertson, "Understanding inverse document frequency: on theoretical arguments for IDF," *J. Doc.*, vol. 60, no. 5, pp. 503–520, 2004.

[48] "Dot products." [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/dot-products-1.html. [Accessed: 07-Oct-2017].

[49] "Document and query weighting schemes." [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/document-and-query-weighting-schemes-1.html. [Accessed: 07-Oct-2017].

[50] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. MIT Press. Cambridge, MA, 1999.

[51] P. Pecina, "An Extensive Empirical Study of Collocation Extraction Methods," in *ACLstudent '05 Proceedings of the ACL Student Research Workshop*, 2005, pp. 13–18.

[52] J. F. da Silva and G. P. Lopes, "A Local Maxima method and a Fair Dispersion Normalization for extracting multi-word units from corpora," in *Proceedings of the 6th Meeting on the Mathematics of Language*, 1999, pp. 369–381.

[53] A. Vilaça, M. Antunes, and D. G. Gomes, "TVPulse: detecting TV highlights in Social Networks," in *10th Conference on Telecommunications Conftele*, 2015.

[54] "Altice Labs." [Online]. Available: http://www.alticelabs.com/en/. [Accessed: 23-Oct-2017].

[55] S. Phuvipadawat and T. Murata, "Breaking news detection and tracking in Twitter," in *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2010, pp. 120–123.

[56] A.-M. Popescu, M. Pennacchiotti, and D. Paranjpe, "Extracting events and event

descriptions from Twitter," in *WWW '11 Proceedings of the 20th International Conference companion on World Wide Web*, 2011, pp. 105–106.

[57] R. Li, K. H. Lei, R. Khadiwala, and K. C. C. Chang, "TEDAS: A Twitter Based Event Detection and Analysis System," in *ICDE '12 Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, 2012, pp. 1273–1276.

[58] N. Alsaedi, P. Burnap, and O. Rana, "Can We Predict a Riot? Disruptive Event Detection Using Twitter," *ACM Trans. Internet Technol. - Spec. Issue Adv. Soc. Comput. Regul. Pap.*, vol. 17, no. 2, 2017.

[59] S. Petrović, M. Osborne, and V. Lavrenko, "Streaming First Story Detection with application to Twitter," in *HLT 2010 - Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010, pp. 181–189.

[60] P. S. Foundation, "Python." [Online]. Available: https://www.python.org/. [Accessed: 04-Sep-2017].

[61] W. Richert and L. P. Coelho, *Building Machine Learning Systems with Python*, Second Edi. Packt Publishing, 2015.

[62] "About Cython." [Online]. Available: http://cython.org/. [Accessed: 04-Sep-2017].

[63] P. S. Foundation, "PyPI - the Python Package Index." [Online]. Available: https://pypi.python.org/pypi. [Accessed: 04-Sep-2017].

[64] "Anaconda." [Online]. Available: https://docs.continuum.io/. [Accessed: 04-Sep-2017].

[65] N. Developers, "Numpy." [Online]. Available: http://www.numpy.org/. [Accessed: 04-Sep-2017].

[66] "Python Data Analysis Library." [Online]. Available: http://pandas.pydata.org/. [Accessed: 04-Sep-2017].

[67] "Introduction." [Online]. Available: http://matplotlib.org/. [Accessed: 04-Sep-2017].

[68] M. Waskom, "seaborn: statistical data visualization." [Online]. Available: https://seaborn.pydata.org/. [Accessed: 04-Sep-2017].

[69] "Scipy." [Online]. Available: https://www.scipy.org/. [Accessed: 04-Sep-2017].

[70] S. D. Team, "About." [Online]. Available: http://www.sympy.org/en/index.html. [Accessed: 04-Sep-2017].

[71] P. Jupyter, "Jupytercon." [Online]. Available: https://jupyter.org/. [Accessed: 04-Sep-2017].

[72] "The R Project for Statistical Computing." [Online]. Available: https://www.r-project.org/. [Accessed: 07-Nov-2017].

[73] "Julia." [Online]. Available: https://julialang.org/. [Accessed: 07-Nov-2017].

[74] "nbviewer: A simple way to share Jupyter Notebooks." [Online]. Available:

http://nbviewer.jupyter.org/. [Accessed: 04-Sep-2017].

[75] "Apache Spark." [Online]. Available: https://spark.apache.org/. [Accessed: 06-Sep-2017].

[76] "JupyterHub." [Online]. Available: https://jupyterhub.readthedocs.io/en/latest/. [Accessed: 05-Oct-2017].

[77] "Google." [Online]. Available: https://www.google.pt/intl/en_pt/about/our-company/. [Accessed: 07-Nov-2017].

[78] "Microsoft." [Online]. Available: https://www.microsoft.com/pt-pt/. [Accessed: 07-Nov-2017].

[79] "IBM." [Online]. Available: https://www.ibm.com/us-en/. [Accessed: 07-Nov-2017].

[80] "Software for complex network." [Online]. Available: https://networkx.github.io/. [Accessed: 07-Sep-2017].

[81] "NetworkX Overview." [Online]. Available: https://networkx.github.io/documentation/networkx-1.9.1/overview.html. [Accessed: 07-Nov-2017].

[82] R. Yu, H. Qiu, Z. Wen, C.-Y. Lin, and Y. Liu, "A Survey on Social Media Anomaly Detection," *ACM SIGKDD Explorations Newsletter*, vol. 18, no. 1, pp. 1–14, 2016.

[83] "scikit-learn Machine Learning in Python." [Online]. Available: http://scikit-learn.org/stable/. [Accessed: 06-Sep-2017].

[84] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[85] "The 2-Clause BSD License." [Online]. Available: https://opensource.org/licenses/BSD-2-Clause. [Accessed: 06-Sep-2017].

[86] A. Morin, J. Urban, and P. Sliz, "A Quick Guide to Software Licensing for the Scientist-Programmer," *PLoS Comput. Biol.*, vol. 8, no. 7, 2012.

[87] "Ubuntu." [Online]. Available: https://www.ubuntu.com/. [Accessed: 07-Nov-2017].

[88] "Debian." [Online]. Available: https://www.debian.org/. [Accessed: 07-Nov-2017].

[89] S. Developers, "Strategies to scale computationally: bigger data." [Online]. Available: http://scikit-learn.org/stable/modules/scaling_strategies.html. [Accessed: 06-Sep-2017].

[90] "Natural Language Toolkit." [Online]. Available: http://www.nltk.org/. [Accessed: 29-Sep-2017].

[91] S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[92] "WordNet." [Online]. Available: https://wordnet.princeton.edu/. [Accessed: 07-Nov-2017].

[93] G. Harrison, *Next Generation Databases NoSQL, NewSQL and Big Data*. Apress, 2015.

[94] “MongoDB Limits and Thresholds.” [Online]. Available: https://docs.mongodb.com/manual/reference/limits/. [Accessed: 05-Oct-2017].

[95] “Couchbase Start a Revolution.” [Online]. Available: https://www.couchbase.com/. [Accessed: 05-Oct-2017].

[96] “MongoDB For Giant Ideas.” [Online]. Available: https://www.mongodb.com/. [Accessed: 05-Oct-2017].

[97] I. MongoDB, “MongoDB Architecture.” [Online]. Available: https://www.mongodb.com/mongodb-architecture. [Accessed: 31-Aug-2017].

[98] MongoDB, “MongoDB Architecture Guide,” *MongoDB White Paper*. 2017.

[99] “Apache Cassandra.” [Online]. Available: http://cassandra.apache.org/. [Accessed: 07-Nov-2017].

[100] “Redis.” [Online]. Available: https://redis.io/. [Accessed: 05-Oct-2017].

[101] “VoltDB.” [Online]. Available: https://www.voltdb.com/. [Accessed: 05-Oct-2017].

[102] “Redis Persistence.” [Online]. Available: https://redis.io/topics/persistence. [Accessed: 05-Oct-2017].

[103] R. Akbani, S. Kwek, and N. Japkowicz, “Applying Support Vector Machines to Imbalanced Datasets,” in *Machine Learning: ECML 2004, 15th European Conference on Machine Learning*, 2004, pp. 39–50.

[104] K. Wang, C. Thrasher, E. Viegas, X. Li, and B. Hsu, “An overview of Microsoft Web N-gram corpus and applications,” in *HLT-DEMO ’10 Proceedings of the NAACL HLT 2010 Demonstration Session*, 2010, pp. 45–48.

[105] “Index of /ptwiki/latest/.” [Online]. Available: https://dumps.wikimedia.org/ptwiki/latest/. [Accessed: 27-Sep-2017].

[106] “API Overview.” [Online]. Available: https://dev.twitter.com/overview/api. [Accessed: 22-Sep-2017].

[107] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia, “Detecting automation of Twitter accounts: Are you a human, bot, or cyborg?,” *IEEE Trans. Dependable Secur. Comput.*, vol. 9, no. 6, pp. 811–824, 2012.

[108] Twitter, “Field Guide.” [Online]. Available: https://dev.twitter.com/overview/api/tweets. [Accessed: 20-Jul-2017].

[109] MongoDB, “Production Notes.” [Online]. Available: https://docs.mongodb.com/manual/administration/production-notes/. [Accessed: 19-Jul-2017].

[110] Couchbase, “Supported Platforms.” [Online]. Available:

https://developer.couchbase.com/documentation/server/4.6/install/install-platforms.html. [Accessed: 19-Jul-2017].

[111] “GlobalInterpreterLock.” [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock. [Accessed: 26-Oct-2017].

Appendix A

```
def _findMaxPath(self, dagNodes, endNode, wikiWordLookup, countWordLookup):
    ## sort positions
    sortPos = sorted(dagNodes)

    for idx in range(1, len(sortPos)):
        for node in dagNodes[sortPos[idx]].values():
            ## compute cost for nodes in pos
            node.cost = 0 if node.bypass \
                          else self._scoreNGram(node, ikiWordLookup, countWordLookup)
            maxCost = 0

            for inNode in node.incomingNodes().values():
                cost = inNode.cost + node.cost
                if cost >  maxCost:
                    maxCost = cost
                    node.maxInNode = inNode

            node.cost += maxCost

    node = endNode.maxInNode
    segs = []

    while node:
        if node.name != 'START" and not node.bypass:
            segs.append((node.name, node.wikiAnchorProb, node.ewikiAnchorProb))

        node = node.maxInNode

    return segs
```

Listing 15: Python iterative implementation of the max path search algorithm.

```
def fit(self):
    ## inittially each element is in a separate cluster
    self.oClusters= {k: k for k in range(self._nCols)}
    self.clusters = defaultdict(set)
    self.edges = defaultdict(list)

    ### compute k nearest neighbours
```

```
    kn = self._computeNeighbours()

    for idx in range(self._nCols - 1):
        for idj in kn[idx]:
            if idj < idx:
                ## has previously been tested
                continue

            if idx in kn[idj]:
                if self.oClusters[idx] != self.oClusters[idj]:
                    cId = min(self.oClusters[idx], self.oClusters[idj])
                    oId = max(self.oClusters[idx], self.oClusters[idj])

                    ## assign to cluster
                    self._assignToCluster([idx, idj], cId)

                    ## update clusters and edges
                    self._update(cId, [idx, idj])

                    ## merge clusters if necessary
                    self._merge(oId, cId)
```

Listing 16: Python implementation of the Jarvis-Patrick variant.

```
def _selectFeatures(self):

    ## use 80% of the data for training and 20% for testing
    xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.20)
    featsLabels = np.array(['seg', 'edge', 'wiki', 'sim', 'df', 'udf', 'rt', 'men', 'rep', 'url', 'tag'])

    forest = RandomForestClassifier(n_estimators=10000, n_jobs=2)
    forest.fit(xTrain, yTrain)
    importances = forest.feature_importances_
    inds = np.argsort(importances)[::-1]

    ## visualize the importances given to each feature
    for idx in range(xTrain.shape[1]):
        print("%2d) %-*s %f" % (idx + 1, 30, featsLabels[inds[idx]], importances[inds[idx]]))
```

Listing 17: Python code used to compute the importance of the features.

```
def _findBestEstimator(self, xTrain, yTrain, xTest, yTest):

    X, y = self._standardizeData(self._data)

    ## use 80% of the data for training and 20% for testing
    xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.20)

    ## choose the best parameters for the classifier
    pRange = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
    pGrid = [{'C': pRange, 'kernel': ['linear']}, {'C': pRange, 'gamma': pRange, 'kernel': ['rbf']} ]
```

```
gs=GridSearchCV(estimator=SVC(), param_grid=pGrid, scoring='accuracy', cv=5, n_jobs=2)
gs = gs.fit(xTrain, yTrain)
clf = gs.best_estimator_

print('Best gridsearch estimator score: {}'.format(gs.best_score_))
print('Best gridsearch estimator parameters: {}'.format(gs.best_params_))

yTrue, yPred = yTest, clf.predict(xTest)
print(classification_report(yTrue, yPred))

## best classifier estimator
return clf
```

Listing 18: Python code used to find the best estimator parameters.

Appendix B

Stickiness Function

The stickiness function $C(s)$, depicted in Equation 9 in Subsection 4.3.1 (and also below for ease of consultation, in Equation A), is used in order to measure the stickiness of segments. The meaning of stickiness in this case can be somewhat related to the notion of cohesion, in that a higher stickiness score, indicates that further splitting the segment would break it as a unit. This function was formalized and is also explained in Section 3.1 of the original paper. The several components of this function are the following:

- function *L*, defined in Equation 14, is used to give moderate preference to longer segments. This is done, given the intuition that longer segments tend to be more informative when compared to shorter ones;
- function *Q(s)* is the probability that segment *s* appears as an anchor text in Wikipedia articles containing it. This is done with the intuition that segments more frequently used as anchors in Wikipedia are more newsworthy;
- function *SCP(s)*, is the Symmetric Conditional Probability and is defined in Equation 8. This function is used in order to try to find segments more semantically meaningful.

The objective of this formulation is to try to obtain segments that are more informative, potentially more newsworthy and also more likely to represent meaningful semantic units. The rationale for this, is that such segments are more suitable for the task of event detection and also more representative of real-world newsworthy events.

$$C(s) = L(s) * e^{Q(s)} * SCP(s) \qquad \text{(A)}$$

$$L(s) = \begin{cases} \dfrac{|s| - 1}{s}, & |s| > 1 \\ 1, & |s| = 1 \end{cases} \qquad (14)$$

Appendix C

Weighting Scheme

The weight of a segment *s*, denoted as $w_b(s,t)$ and computed as depicted in Equation 10 in Subsection 4.3.2 (and also below for ease of consultation, in Equation B), is used in order to rank segments according to this weight and retain only the top K of these for further processing. This function was formalized and is also explained in Section 3.2 of the original paper. The several components of this function are the following:

- function $P_b(s,t)$ is the bursty probability of a segment and is defined in Equation 15. In this equation $f_{s,t}$ denote the frequency of segment *s* within time window *t* (i.e. the number of tweets created in *t* that contain *s*) and *S* denotes the sigmoid function. In this case the probability of observing $f_{s,t}$ is modeled by a binomial distribution, see Equation 16, which in turn can be approximated by a Gaussian distribution, see Equation 17, given that N_t (i.e. the number of tweets created within *t*) is very large. Therefore *E[s|t]* and $\sigma[s|t]$ denote the mean and the standard deviation of this normal approximation and are computed respectively as, $E[s|t] = N_t * p_s$ and $\sigma[s|t] = \sqrt{N_t * p_s * (1 - p_s)}$. The probability that a tweet contains segment *s* in a random time window *t*, denoted as p_s, is computed using Equation 18. *L* denotes the number of time windows *t* containing segment *s* in this equation. This is done with the intuition that segments occurring more frequently than usual (i.e. bursty segments), may be an indication of an event being discussed;
- $log(u_{s,t})$ denotes the *log* (where *log* stands for the *log* function) value of the user support (i.e. the number of distinct users who created tweets containing segment *s* within time window *t*), denoted as $u_{s,t}$. This as to do with the intuition that bursty segments used by more users (i.e. more users creating tweets that contain the segment) may have a higher chance to be related to events. The logarithm of the user frequency however is taken instead of the raw user frequency. This is done in order to try to prevent bursty segments with higher user frequencies from being ranked too high while still allowing bursty

segments with moderate user frequency to be ranked relatively higher than segments with a lower user frequency.

$$w_b(s,t) = P_b(s,t) * \log(u_{s,t}) \tag{B}$$

$$P_b(s,t) = \begin{cases} 1, & f_{s,t} \geq E[s|t] + 2 * \sigma[s|t] \\ S(10 * \dfrac{f_{s,t} - (E[s|t] + \sigma[s|t])}{\sigma[s|t]}), & E[s|t] + 2 * \sigma[a|t] > f_{s,t} > E[s|t] \end{cases} \tag{15}$$

$$P(f_{s,t}) = \binom{N_t}{f_{s,t}} * p_s^{f_{s,t}} * (1 - p_s)^{N_t - f_{s,t}} \tag{16}$$

$$P(f_{s,t}) \sim N(N_t * p_s, N_t * p_s * (1 - p_s)) \tag{17}$$

$$p_s = \frac{1}{L} * \sum_{t=1}^{L} \frac{f_{s,t}}{N_t} \tag{18}$$

Appendix D

Similarity Function

In order to group related segments together, a measure of their similarity must be computed. This similarity measure denoted as $sim_t(s_a,s_b)$, is computed as depicted in Equation 19. This function was formalized and is also explained in Section 3.3 of the original paper. The several components of this function are the following:

- the function $w_t(s,m)$, functions as a measure of the importance or weight of sub-time window m to segment s and is computed as depicted in Equation 20. In this equation, $f_t(s,m)$ denotes the tweet frequency of segment s in sub-time window m. This is done in order to take the temporal frequency patterns of the segments into account;
- the function $sim(T_1,T_2)$, measures the similarity between segments by using the content of their associated tweets. In this function, T_1 and T_2 are the pseudo documents that result from the concatenation of all tweets in $T_t(s_a,m)$ an $T_t(s_b,m)$ (tweets contained in all sub-time windows) respectively and transformed into the TF-IDF scheme. The function $sim(T_1,T_2)$, denotes the cosine similarity measure applied to T_1 and T_2. This is done in order to take the context associated to the segments into account, when computing their similarity.

$$sim_t(s_a, s_b) = \sum_{m=1}^{M} w_t(s_a, m) * w_t(s_b, m) * sim\big(T_t(s_a, m), T_t(s_b, m)\big) \quad (19)$$

$$w_t(s, m) = \frac{f_t(s, m)}{\sum_{m'=1}^{M} f_t(s, m')} \quad (20)$$

Given these two factors, two segments s_a and s_b, are expected to be similar and therefore belong to the same event, if both their associated context (tweets content) and frequency patterns along the M sub-time windows are also similar. Dissimilar context may suggest that the segments belong to distinct events, while inconsistent frequency patterns may suggest that the segments refer to similar events occurring at different times (e.g. two football matches occurring in the same day).

Appendix E


```
{
"in_reply_to_status_id_str":null,
"in_reply_to_status_id":null,
"created_at":"Wed Aug 24 22:59:59 +0000 2016",
"in_reply_to_user_id_str":null,
"source":"<a href='http://twitter.com/download/iphone' rel='nofollow'>Twitter for iPhone</a>",
"retweet_count":0,
"retweeted":false,
"geo":null,
"filter_level":"low",
"in_reply_to_screen_name":null,
"is_quote_status":false,
"id_str":"768583664632532992",
"in_reply_to_user_id":null,
"favorite_count":0,
"id":768583664632532992,
"text":"Há coisas q me tiram mm do sério, enfim",
"place": {
        "country_code":"PT",
        "country":"Portugal",
        "full_name":"Portimão, Portugal",
        "bounding_box":{
                "coordinates":[[[-8.68,37.1086],[-8.68,37.279918],[-8.48,37.279918],[-
8.48,37.1086]]],
                "type":"Polygon"
        },
        "place_type":"city",
        "name":"Portimão",
        "attributes":{},
        "id":"e68b57affea10f1d",
        "url":"https://api.twitter.com/1.1/geo/id/e68b57affea10f1d.json"
},
"lang":"pt",
"favorited":false,
"coordinates":null,
"truncated":false,
"timestamp_ms":"1472079599899",
"entities":{
        "urls":[],
```

```
        "hashtags":[],
        "user_mentions":[],
        "symbols":[]
},
"contributors":null,
"user":{
        "utc_offset":null,
        "friends_count":252,
        "profile_image_url_https":"https://pbs.twimg.com/profile_images/762714628/FuGflf7K_n
ormal.jpg",
        "listed_count":2,
        "profile_background_image_url":"http://pbs.twimg.com/profile_background_images/lh_G
SvhY.jpg",
        "default_profile_image":false,
        "favourites_count":12091,
        "description":null,
        "created_at":"Fri Mar 28 21:20:14 +0000 2014",
        "is_translator":false,
        "profile_background_image_url_https":"https://pbs.twimg.com/profile_background_image
s/lh_GSvhY.jpg",
        "protected":false,
        "screen_name":"biiasiilva98",
        "id_str":"2442928269",
        "profile_link_color":"DD2E44",
        "id":2442928269,
        "geo_enabled":true,
        "profile_background_color":"000000",
        "lang":"pt",
        "profile_sidebar_border_color":"000000",
        "profile_text_color":"000000",
        "verified":false,
        "profile_image_url":"http://pbs.twimg.com/profile_images/FuGflf7K_normal.jpg",
        "time_zone":null,
        "url":"http://Instagram.com/bialexandranunesilva",
        "contributors_enabled":false,
        "profile_background_tile":true,
        "profile_banner_url":"https://pbs.twimg.com/profile_banners/2442928269/1471941120",
        "statuses_count":54051,
        "follow_request_sent":null,
        "followers_count":1142,
        "profile_use_background_image":true,"
        default_profile":false,
        "following":null,
        "name":"nita",
        "location":"Oliveira de Azeméis ",
        "profile_sidebar_fill_color":"000000",
        "notifications":null
}
}
```


Listing 19: Example of a collected tweet.

```
{
"in_reply_to_status_id_str":null,
"in_reply_to_status_id":null,
"created_at":"Wed Aug 24 22:59:59 +0000 2016",
"in_reply_to_user_id_str":null,
"source":"<a href='http://twitter.com/download/iphone' rel='nofollow'>Twitter for iPhone</a>",
"retweet_count":0,
"retweeted":false,
"geo":null,
"filter_level":"low",
"in_reply_to_screen_name":null,
"is_quote_status":false,
"id_str":"768583664632532992",
"in_reply_to_user_id":null,
"favorite_count":0,
"id":768583664632532992,
"text":"Há coisas q me tiram mm do sério, enfim",
"place_country_code":"PT",
"place_country":"Portugal",
"place_full_name":"Portimão, Portugal",
"place_bounding_box_coordinates":[[[-8.68,37.1086],[-8.68,37.279918],[-8.48,37.279918],[-
8.48,37.1086]]],
"place_bounding_box_type":"Polygon"
place_place_type":"city",
place_name":"Portimão",
place_attributes":{},
place_id":"e68b57affea10f1d",
place_url":"https://api.twitter.com/1.1/geo/id/e68b57affea10f1d.json"
"lang":"pt",
"favorited":false,
"coordinates":null,
"truncated":false,
"timestamp_ms":"1472079599899",
"entities_"urls":[],
"entities_hashtags":[],
"entities_user_mentions":[],
"entities_symbols":[]
"contributors":null,
"user_utc_offset":null,
"user_friends_count":252,
"user_profile_image_url_https":"https://pbs.twimg.com/profile_images/FuGflf7K_normal.jpg",
"user_listed_count":2,
"user_profile_background_image_url":"http://pbs.twimg.com/profile_background_images/lh_GSv
hY.jpg",
"user_default_profile_image":false,
"user_favourites_count":12091,
"user_description":null,
"user_created_at":"Fri Mar 28 21:20:14 +0000 2014",
"user_is_translator":false,
"user_profile_background_image_url_https":"https://pbs.twimg.com/profile_background_images/l
h_GSvhY.jpg",
"user_protected":false,
```

```
"user_screen_name":"biiasiilva98",
"user_id_str":"2442928269",
"user_profile_link_color":"DD2E44",
"user_id":2442928269,
"user_geo_enabled":true,
"user_profile_background_color":"000000",
"user_lang":"pt",
"user_profile_sidebar_border_color":"000000",
"user_profile_text_color":"000000",
"user_verified":false,
"user_profile_image_url":"http://pbs.twimg.com/profile_images/762714624408117248/FuGflf7K_
normal.jpg",
"user_time_zone":null,
"user_url":"http://Instagram.com/bialexandranunesilva",
"user_contributors_enabled":false,
"user_profile_background_tile":true,
"user_profile_banner_url":"https://pbs.twimg.com/profile_banners/2442928269/1471941120",
"user_statuses_count":54051,
"user_follow_request_sent":null,
"user_followers_count":1142,
"user_profile_use_background_image":true,"
"user_default_profile":false,
"user_following":null,
"user_name":"nita",
"user_location":"Oliveira de Azeméis ",
"user_profile_sidebar_fill_color":"000000",
"user_notifications":null
}
```


Listing 20: Example of a tweet after flattening the hierarchy.


```
{
        "db" : "dumpdb",
        "collections" : 1,
        "views" : 0,
        "objects" : 28935607,
        "avgObjSize" : 2870.4877125957646,
        "dataSize" : 83059304350,
        "storageSize" : 35227975680,
        "numExtents" : 0,
        "indexes" : 3,
        "indexSize" : 1369899008,
        "ok" : 1
}
```


Listing 21: Database statistics. All size values are presented in bytes.